Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

DESIGN ALTERNATIVES FOR RECURSIVE DIGITAL FILTERS USING ON-LINE ARITHMETIC

J. S. Fernando

November 1993
CSD-930040

UNIVERSITY OF CALIFORNIA
Los Angeles

Design Alternatives for Recursive Digital Filters Using On-Line Arithmetic

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

John Susantha Fernando

1993

Table of Contents

List of Figures

List of Tables

List of Symbols

a, b, c	coefficients in linear expression
m	number of bits used to represent coefficients in 2's complement
d	number of digits in input and output samples
f	number of fraction bits used in implementation
i, j, k	iteration or digit indices
n	sample number
N	order of recursion
r	radix (also used for magnitude of a complex number)
s	sum of maximum magnitudes of coefficients
u, v	FIR inputs
$w[j]$	residual in digit-recurrence algorithm
x, y	sample values of input, output
$\overline{z}, \underline{z}$	bounds of Z_j
R	absolute value of a complex number
R_j	recoder output digit
S, C	sum, carry
X, Y, U	digits of words denoted x, y, u
Z_j	multiplier output digit
t_j, s_j	transfer digit, sum digit
t_{buf}	delay (ns) of fan-out buffer for digit multiplexers,
t_{mux}	multiplexer delay (ns)
t_{fa}	delay of a full adder (ns)
t_{ha}	delay of a half adder (ns)
t_{clk}	clock period (ns)
t_{clkQ}	clock-to-output delay for flip-flops (ns)
t_{set}	set-up time for flip-flops (ns)
t_{cpa}	delay caused by 4-bit CPA
β	number of fraction bits used for digit selection
δ	on-line delay (clocks)
δ_i, δ_j	latency (clocks) between $y(n)$ and $y(n-i)$, $y(n)$ and $y(n-j)$
δ_{imp}	on-line delay of an implementation (clocks)
γ	factor by which SL reduces the amplitude bound of zero-input limit cycles
μ	maximum rate of an array (Msamples/sec)
ρ	maximum value of a digit in a redundant number system
$\underline{\xi}, \overline{\xi}$	bounds of $w[j] - Y_j$ or $w[j] - D_j r^{d-1}$

List of Acronyms

CM	Combined Multiplier
CPA	Carry Propagate Adder
CSA	Carry Save Adder or Carry Save Array
DS	Digit Stage or Dynamic Scaling
FA	Full Adder
FIR	Finite Impulse Response
HA	Half Adder
IIR	Infinite Impulse Response
LSD	Least Significant Digit
MA	Multiply-Add
MSD	Most Significant Digit
RU	Recursive Unfolding
SM	Separate Multiplier
SL	Scattered Lookahead
WM	Word Module

ACKNOWLEDGMENTS

I like to thank the numerous individuals and institutions for their contributions to my education. I owe much to my parents for fostering a love for learning and for the opportunities they provided. I am grateful to the dedicated staff of my high school, St. Joseph's, and also to the University of Sri Lanka for a state-funded undergraduate education. I am also grateful to the University of California, Los Angeles, for the educational opportunities it has provided.

For their encouragement and support I thank my friends and colleagues: Joe Miro-Julia, Paul Tu, Raffi Dionysian, James Liu, Marianne Louie, John Harding, Edwin Tisdale and Dinh Le. Thanks are also due to Prof. Tomas Lang, Verra Morgan, and others too numerous to mention.

I take great pleasure in expressing my gratitude and thanks to my advisor Prof. Miloš Ercegovac for many things. First, for introducing me to computer arithmetic, on-line in particular, and to the dissertation topic. Second, for advice and guidance in research. Third, for having me as a research assistant. Finally, for being available for consultation, often without appointment, during working hours.

I am deeply grateful and thankful to my wife Rasanjali for her patience and encouragement, and for readily assuming much of the burden of supporting us during this endeavor.

This research has been supported in part by NSF grant No. MIP-8813340 *Composite Operations Using On-Line Arithmetic for Application-Specific Parallel Architectures: Algorithms, Design and Experimental Studies* and by the State of California MICRO Program and Rockwell International.

VITA

1978	B.Sc. in Engineering (Honors) University of Sri Lanka, Moratuwa
1983	M.S. in Electrical Engineering University of Texas at Austin

PUBLICATIONS

John S. Fernando and Miloš D. Ercegovac,
Conventional and On-Line Arithmetic Designs for High-Speed Recursive Digital Filters, VLSI Signal Processing, V, IEEE Press, K. Yao, R. Jain, W. Przytula, J. Rabaey Eds., 1992, pp. 81–90.

John S. Fernando and Miloš D. Ercegovac,
On-Line Arithmetic Modules For Recursive Digital Filters, Proc. 26th Annual Asilomar Conference on Signals, Systems and Computers, 1992, pp. 681–685.

ABSTRACT OF THE DISSERTATION

Design Alternatives for Recursive Digital Filters Using On-Line Arithmetic

by

John Susantha Fernando
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 1993
Professor Miloš Ercegovac, Chair

Design alternatives for fixed-point recursive computations of a second-order direct form filter using on-line arithmetic are analyzed. On-line implementations consume inputs and produce outputs digit serially, beginning with the most significant digit. Digit-level pipelining of on-line filters result in high sampling rates independent of word length. Designs are implemented in a gate array technology to compare performance and costs. Since technology, word length, and function are identical for all designs, the impact of different arithmetic algorithms on performance and cost is easily compared. On-line modules can be used unmodified to implement recursions with scattered lookahead (SL), a transformation that speeds up both on-line and conventional designs. With 2 levels of SL, arrays composed of on-line multiply-add modules implemented in a 0.7-micron gate array, deliver 128Msamples/second. For words exceeding 12 bits, such on-line arrays are faster than a conventional design implemented in an identical gate array. For 16-bit words, on-line arrays are 20% faster. Theoretical bounds derived show that undesirable limit cycle oscillations are reduced in amplitude for most useful filters when SL is applied. The Dynamic Scaling (DS) scheme is proposed to eliminate all undesirable oscillations. Implementation shows that the DS scheme is twice as cost effective as the precision extension method of eliminating oscillations. For single modules with 8-bit I/O words, the DS scheme has a rate 80% higher than the precision extension method and is 13% smaller. The DS scheme requires no modifications of on-line modules.

CHAPTER 1

Introduction

The usual method of speeding up a computation exploits inherent parallelism. Thus, several computational units may be used, and each unit can be pipelined to achieve high clock rates. For a recursive computation of a sequence of values, the rate cannot be increased by simple pipelining, particularly when conventional (right to left evaluation) arithmetic is used. To extract parallelism, and speed up the solution of a general class of linear recurrence equations, a transformation technique called recursive doubling was used by Kogge [Kog73], [Kog81]. The technique transforms a function into two subfunctions of equal complexity, allowing parallel computation of the subfunctions. Successive application of the transform doubles the number of subfunctions. Recursive doubling solutions for second-order linear recurrences were known as early as 1853 to J.J. Sylvester [Kog73]. Two methods of increasing the throughput (rate) of recursive and non-recursive digital filters were proposed by Hayashi et al. [HDSH86]. For a given transfer function, $H(z)$, an N-fold rate increase is obtained by transforming $H(z)$ into component transfer functions. A similar transformation called scattered lookahead (SL) was proposed by Parhi and Messerschmitt to speed-up first-order and second-order transfer functions [PrMs87a], [PrMs87b], [Ms88]. SL allows limited pipelining of the recursive loop in conventional implementations [PrHt88], [HtPr92]. The transformations referred to above achieve speed-up by exposing parallelism at the algorithm level.

Independent of any algorithmic transformations, the rate of recursive computations can be increased at the arithmetic level by using the on-line or most-significant-digit-first (MSDF) method of computation [Erc84], [ErLn88], [ErLn89], or by using the MSDF systolic array approach [KnMc88], [KnMc89], [KnMc90], [KnMc91]. On-line implementations are described in [Bra89a] and [Bra89b].

On-line Arithmetic

On-line arithmetic algorithms, proposed by Trivedi and Ercegovac [TrEc77], are a class of digit-serial algorithms that compute the most significant digit first (MSDF). On-line algorithms have been developed for the basic operations like addition, multiplication, division and square root [ErLn88], [Tu90]. On-line algorithms have also been developed for more complex operations like computing rotation factors, solving classes of linear equations, and singular value decomposition [TuEr91].

Characteristics of on-line algorithms are summarized below.

- Inputs are consumed and outputs are produced, digit by digit, beginning with the most significant.
- The output digits are selected from a redundant digit set. The input digits are usually from the same digit set. This allows the cascading of on-line units.
- The first output digit becomes available only after a few input digits have been consumed. This delay is called the *on-line delay* and is an important characteristic of an on-line algorithm.
- A systematic methodology exists for developing on-line algorithms [ErLn88].

Two essential components of an on-line algorithm are the *residual recurrence* evaluation and the output *digit selection.* In each iteration, or step, input digits are consumed, a new *residual* (an internal state) is computed and an output digit is selected. To enable fast computation, the residual is usually maintained in redundant form (carry-save or signed-digit). Benefits of on-line computation are:

- Digit-level parallelism, which allows the next computation to begin in another module after the on-line delay.
- Low interconnection overhead, because routing between on-line units is on a digit basis.
- Clock rate independent of word length.
- Variable precision operation: at step j, the on-line result produced corresponds to the j most significant digits of the conventional result.

In general, due to digit-level parallelism, on-line algorithms can speed-up computations consisting of a sequence of basic operations. Since recursive computations are an infinite sequence of a few basic operations, the on-line approach is well suited for such computations.

1.1 Problem Statement

The central problem in this dissertation is how to use on-line arithmetic to speed-up computation of a second-order linear recurrence given by (1.1) or (1.2). Considering second-order recurrences is sufficient because higher order filters are implemented by compositions of second-order sections [PrMn88].

$$y(n) = a \cdot y(n-1) + b \cdot y(n-2) + c \cdot x(n) \tag{1.1}$$

or

$$y(n) = a \cdot y(n-1) + b \cdot y(n-2) + u(n) \tag{1.2}$$

The term $u(n)$, also called the finite impulse response (FIR) term, is a non-recursive linear function of $x(n)$, $x(n-1)$, and $x(n-2)$. These expressions correspond to direct form filter structures. The direct form filter structures have been shown to be best suited for high sampling rates [Sam88]. Unfortunately, the direct form filter suffers from undesirable non-linear oscillations. The oscillations must be eliminated without compromising much of the sampling rate and with minimum added cost.

1.2 Research Objectives

The few designs and implementations described in recent publications cannot be compared because of differences in technology, word length, coefficient magnitude, order of output function, and features like testability. Key questions remain:

1. Given a performance level or cost constraint, what is the most appropriate architecture for implementing a second-order direct form recursive digital filter?
2. At what word length does on-line arithmetic payoff, compared to conventional?
3. How does the scattered lookahead transformation, used to increase sampling rate, affect non-linear oscillations?
4. How can non-linear oscillations be eliminated from on-line implementations?

Currently, the only method of eliminating non-linear oscillations in on-line modules is to extend working precision [Bra89b]. The method does not reduce the clock rate, but requires a costly increase of working precision. A more cost-effective solution is developed in Chapter 8.

1.3 Dissertation Outline

The emphasis in this dissertation is on the recursive part of the computation, since the FIR part of the computation is easily performed at the desired rate. Simple schemes based on on-line multipliers, on-line adders, and on-line recoders, are analyzed in Chapter 2. These schemes are easy to develop but have excessive on-line delay. On-line composite modules [ErLn88], which perform the computation as a whole, are developed in Chapter 3. An expression for the on-line delay is derived for a N-order linear recurrence. Algorithms with small on-line delay are implemented. Composite modules are more cost-effective and faster than the simple schemes. On-line multiply-add (MA) modules, which can be cascaded to realize (1.1) or (1.2), are developed in Chapter 4. MA modules are faster than composite modules. Chapter 5 describes two types of multi-module maximum rate arrays based on composite and MA modules. The rate of the array depends on the clock rate of the basic module, and the size of the array depends mainly on the word length.

Composite modules and MA modules with different digit sets and different levels of pipelining are implemented in a LCA10000[1] Series Compacted Array[2] [LSI87]. Schematic capture and simulation of designs was done using Workview[3] CAD tools. Performance is measured by the fastest clock rates achievable in simulation. The number of *equivalent* gates is used as a cost measure. Each gate is considered to be equivalent to two n-channel and two p-channel transistors. Arithmetic models and netlists of the on-line implementations are used for functional simulation to verify results.

Chapter 6 discusses algorithmic transformations, sometimes called lookahead schemes, that produce higher sampling rates. Applied to (1.1) or (1.2), such transformations produce equivalent linear recurrences of higher order. Scattered lookahead (SL) is one such transformation proposed to speed-up conventional implementations [PrMs87a], [PrHt88]. Arrays of on-line modules using SL are designed. A new conventional design with two levels of SL, described in appendix

[1]LCA10000 is a gate array using a 1.5-Micron HCMOS technology.

[2]Compacted Array is a trademark of LSI Logic Corporation.

[3]Workview is a trademark of VIEW*logic* Systems Inc..

A, is implemented to find the word length at which on-line implementations outperform the conventional. The effect of SL on the frequency response of two filters is shown in appendix B.

Chapters 7 and 8 investigate non-linear oscillations. Overflow oscillations are caused by limited precision, and zero-input limit-cycle oscillations are caused by roundoff errors [Sam88]. Absolute value bounds for the magnitude of zero-input limit cycle oscillations, similar to [LnTr73], are derived for the SL transformation in Chapter 7. It is shown that SL reduces the bound for the most useful filters. Chapter 8 describes the proposed dynamic scaling scheme which eliminates oscillations in filters composed of on-line modules. Implementation shows that it is twice as cost effective as the precision extension scheme. Chapter 9 compares the designs developed here with others. The dissertation concludes with Chapter 10 which discusses the applicability of the results to other technologies and directions for further research.

1.4 Related Work

Recent publications have shown that digit-serial computation, using the most significant digit first (MSDF), can produce high sampling rates for recursive digital filters. The MSDF approach reduces the recursive bottleneck form the word level to the digit level. Lookahead transformations can be used for doubling or quadrupling sampling rates. The related work is outlined below.

1. In an overview of on-line arithmetic, Ercegovac describes an on-line algorithm computing $y = ax + b$ for solving an mth order linear recurrence system [Erc84].

2. Ercegovac and Lang describe MSDF and on-line approaches for the design of recursive digital filters [ErLn89].

3. Knowles et al. use a MSDF approach for digital filtering, implementing a systolic array for a first-order and second-order recursions [KnMc88], [KnMc89], [KnMc90], [KnMc91].

4. Parhi and Hatamian describe a high speed conventional implementation which uses two levels of scattered lookahead [PrHt88], [HtPr92].

5. Brackert et al. design and implement an on-line multiply-add (MA) module and a direct form II filter composed of MA modules [Bra89a]. This is the first application of on-line arithmetic to implement high speed filters.

6. Cha describes in detail a CMOS chip implementing a direct form II second-order structure using four on-line MAs with programmable coefficients [Cha91]. The implementation uses the MA design described in [Bra89a].

7. McCanny et al. describe a 40MHz IIR filter chip for second-order recursions using a MSDF systolic array architecture with one level of SL [KnMc91]. The architecture is similar to that proposed by Knowles et al. [KnMc88].

Recent implementations of recursive filters fall into three categories: on-line, MSDF systolic arrays, and conventional. On-line designs are described in [Bra89a], [Bra89b], [Cha91], and [FnEr92b]. The MSDF systolic arrays are described in [KnMc88], [KnMc90], [KnMc91] and [LFH90]. Conventional implementations are described in [PrHt88], [HtPr92], and [FnEr92a]. Chapter 9 evaluates some of these implementations in more detail.

1.5 Contributions

Answers to the questions posed in Section 1.2 have been obtained. The main contributions are:

1. Design and implementation of composite on-line modules. Composite on-line modules compute the recursion expressed by (1.1) or (1.2) as a whole. Composite modules have not been designed and implemented before.

2. Design and implementation of a MA module (much faster than [Bra89a]). MA modules are used to implement the recursion as a series of multiply-adds.

3. Development of word module arrays and digit stage arrays based on composite modules and MA modules.

4. New conventional design to implement (1.2) with two levels of scattered lookahead. The design uses a carry-free recursive loop to speed-up computation. It is suited for multiplication of words of about 16 bits or more, depending on the technology. Comparison shows that for precisions of more than 12 bits, the on-line designs are faster.

5. Derivation of new absolute bounds for zero-input limit cycles for the scattered lookahead case. For second-order recursions with distinct complex roots, SL reduces the amplitude bound of zero-input limit cycle oscillations

for most recursions. The recursions for which the amplitude is reduced by SL are those most used in practice. The bounds are verified by simulation.

6. The dynamic scaling (DS) scheme for on-line designs is proposed and implemented. The DS scheme eliminates overflow oscillations and zero-input limit cycle oscillations at half the cost of the precision extension method proposed by Brackert [Bra89b]. Moreover, the DS scheme requires no modification of on-line modules used in the computation.

The contributions can be summarized as follows. Given word length and other constraints, this research shows how to determine the appropriate design: i.e., conventional or on-line, type of on-line module or array, whether to use DS. In addition, theoretical bounds are derived for the absolute bound on limit cycles for second-order sections with SL.

CHAPTER 2

Schemes Based on Multipliers, Recoders and Adders

This chapter describes two simple schemes for realizing Equation (1.1) using a most-significant-digit-first (MSDF) multiplication algorithm described in [Erc87]. This algorithm produces the product in a 2's complement form without using a carry propagate adder and without extra delay. Analysis of these simple schemes provides the insight and the motivation for faster designs described in later chapters.

The first section of this chapter describes the multiplication algorithm, the basis for all modules described in this dissertation. Section 2.2 develops a recurrence-based recoder to convert the over-redundant digit produced by the multiplication. The Separate Mutiplier (SM) scheme and the Combined Multiplier (CM) scheme are described in Sections 2.3 and 2.4. Performance (sampling rate) estimates for each scheme show how performance is influenced by the design, the range of coefficients and the choice of radix and digit sets.

The SM scheme, described in Section 2.3, is the most intuitive implementation of (1.1). The scheme uses three separate MSDF sequential multipliers and two on-line adders as shown in Figure 2.1. This scheme is a result of directly mapping simple arithmetic operations to units which execute these operations in a MSDF manner. The recoders shown in Figure 2.1 convert the output of the multipliers into a digit-set acceptable to the adders. The choice of this scheme as the starting point of the analysis is also justified by the following. First, the use of dedicated units for multiplication, recoding and adding contributes to efficient and fast designs for each unit. Second, having chosen the scheme, a delay analysis reveals the modifications necessary for improved designs.

The CM scheme, described in Section 2.4, combines the three multipliers and computes in fewer clocks than the SM scheme. The combining of the three multiplications into one carry-save adder array eliminates the delays caused by the addition required when the mutipliers are separate. The combining also lowers the hardware required for the CM scheme. However, the larger carry-save adder array reduces the clock rate of the CM scheme.

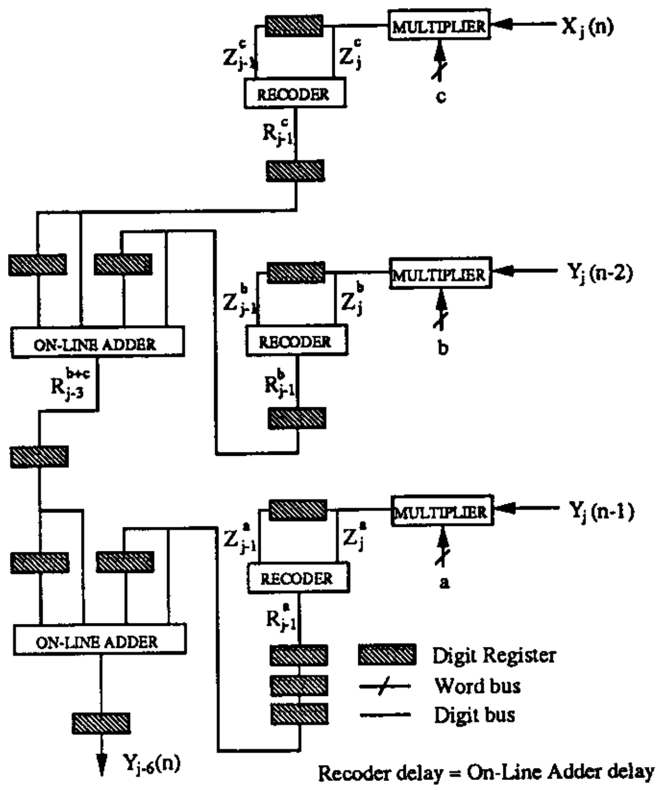


Figure 2.1: Scheme Using On-Line Units

2.1 MSDF Multiplication Algorithm

The selected multiplier uses a MSDF residual recurrence algorithm described in [Erc87]. All multiplications in (1.1) have a constant coefficient as the *multiplicand.* The *multiplier* is in an on-line form. The essential features of the multiplication algorithm are summarized below.

- The residual recurrence uses the digits of the multiplier from left to right, producing the most significant digit of the result first. This is well suited to the filter application, because the computation of $y(n+1)$ may begin in another module as soon as the most significant digit of $y(n)$ is produced.

- The partial product is maintained as a signed-digit integer part and a carry-save fraction part. The carry-save form enables a fast recurrence step because carry propagation is avoided in the recurrence loop. However, each iteration produces an over-redundant signed-digit as the output of the multiplier.

- It is assumed, without loss of generality, that the *mutiplicand* is represented in two's complement, and that the *multiplier* is represented in a radix-4 signed-digit system with the digit set $\{-\rho,\ldots,\rho\}$ where $\rho \in \{2,3\}$. The algorithm is easily modified for a different radix and digit set. Changing the radix or the digit set produces implementations with different performance and cost.

- The *mutiplicands* in all three multipliers are constant coefficients, a, b, c, as defined in (1.1). This allows multiples of the coefficients to be precomputed, permitting the use of digit sets with $\rho > 2$. Stability of the second-order filter requires $|a| < 2$ and $|b| < 1$ [PrMn88].

Residual Recurrence for Multiplication

Consider the multiplication $x \cdot y$, where the multiplicand x is represented in 2's complement and the multiplier y is represented in radix-4 signed-digit form (x and y are used here in a generic sense and do not refer to Equation (1.1)). The values x and y can be expressed as follows.

$$x = -2X_{-1} + \sum_{i=0}^{m-2} X_i 2^{-i}, \quad X_i \in \{0,1\}, \quad |x| < 2 \tag{2.1}$$

$$y = \sum_{i=0}^{d-1} Y_i 4^{-i}, \quad Y_i \in \{-\rho,\ldots,\rho\}, \quad |y| < 1 \tag{2.2}$$

The multiplication algorithm uses a recurrence which produces the sequence of over-redundant signed-digits Z_j and partial products $w[j]$ specified by:

$$w[0] = 0 \tag{2.3}$$

$$w[j+1] = 4\mathit{fraction}(w[j] + xY_j), \quad j = 0,\ldots,d-1 \tag{2.4}$$

$$Z_j = \mathit{integer}(w[j] + xY_j) \tag{2.5}$$

A fast implemention of this recurrence maintains the partial product $w[j]$ in carry-save form, as shown below. The carry-save form allows the use of a carry-save adder which eliminates carry propagation in the recurrence loop.

$$w[j] = S[j] + C[j] \tag{2.6}$$

$$w[j] \geq 0 \tag{2.7}$$

Figure 2.2 shows the residual recurrence step, and Figure 2.3 shows how $Z_j \in \{-4,\ldots,10\}$ is computed by a small CPA for $r = 4$ and $\rho = 2$. The range of Z_j for different radices, digit sets and coefficient bounds are discussed later (Table 2.1).

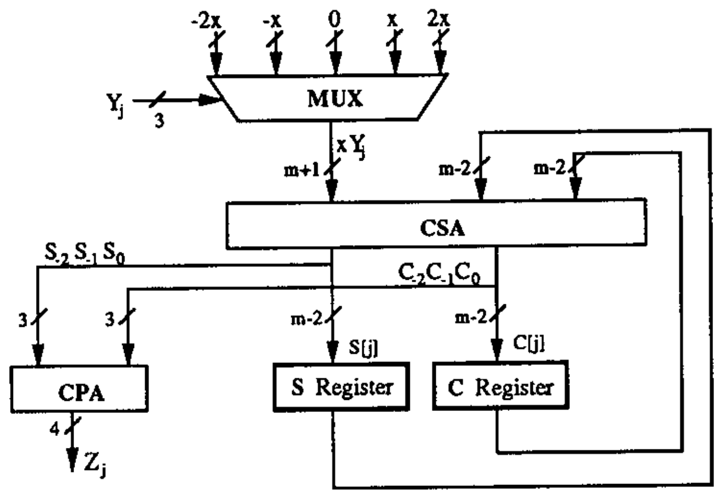


Figure 2.2: Recurrence for Multiplication

O O X X . X ... X O O	4S[j-1]
O O X X . X ... X O O	4C[j-1]
S S X X . X ... X X X	xY_j
S_{-3} S_{-2} S_{-1} S_0 . X ... X X X	S[j]
O C_{-2} C_{-1} C_0 . X ... X X O	C[j]
X X X X .	Z_j

Figure 2.3: Multiplier Output (Z_j) Computation

Implementation Schemes for Multiplication

The critical path of the second-order filter shown in Figure 2.1 consists of a multiplier, a recoder, and an on-line adder. Except for the multiplier, all other units can be pipelined further. Speed of multiplication is limited by the the residual recurrence loop which cannot be pipelined. Therefore, the first stage of the filter is the slowest stage. The radix and the digit-set chosen for the multiplier influence cost and performance of the filter.

The three multiplication schemes shown in Figure 2.4 illustrate the cost and performance trade-offs. Scheme M1 has the minimum clock period because the CPA for producing Z_j is included in the second stage. Scheme M2 has one stage and the slowest clock period. Scheme M3 is a compromise where the CPA for producing Z_j is pipelined into two stages with balanced delays. Scheme M3 is useful only if the CPA delay is larger than the delay for the residual recurrence. The delays $t1_{mul}$ and $t2_{mul}$, for the first stage of schemes M1 and M2, are:

$$t1_{mul} \quad = \quad t_{buf} + t_{mux} + t_{fa} + t_{clkQ} + t_{set}$$

$$t2_{mul} = t_{buf} + t_{mux} + t_{fa} + t_{cpa} + t_{clkQ} + t_{set}$$

where

t_{buf}	Delay of fan-out buffer for the digit multiplexers,
t_{mux}	Multiplexer delay,
t_{fa}	Delay caused by a full adder,
t_{clkQ}	Clock-to-output delay for flip-flops,
t_{set}	Set-up time for flip-flops.
t_{cpa}	Delay caused by 4-bit CPA; $t_{cpa} = t_{mux} + t_{fa} + t_{ha}$

The speed of the multiplier decreases with the size of the digit set, since larger multiplexers have larger values of t_{mux}. For small radices, 2 or 4, the change in t_{mux} caused by the digit set is small and depends on the technology. For example, in LCA10000 technology, the delays for the first five components listed above are (in same order), 1.5ns, 2.5ns, 2.3ns, 1.7ns and 0.8ns [LSI87]. If the 5-to-1 multiplexer is replaced by a 3-to-1 multiplexer, required for a radix-2 scheme, the multiplexer delay drops from 2.5ns to 1.3ns. This represents a speed-up of 13.6% with respect to the radix-4 scheme. Note that delays associated with wire lengths were not included, since such delays depend on the layout. If wire length delays are included, the difference between radix-4 and radix-2 speeds may be smaller.

2.2 Recoder Design

For a computation that produces an over-redundant result, a recoder is required to represent the result in an acceptable digit set. The multiplication algorithm described previously and the addition of signed-digits are examples of such computations. Recoding can be performed in two ways:

1. A series of successive recodings, producing intermediate digit sets that become smaller. Each recoder reduces the digit set by a relatively small amount. An example of such a recoder is shown in Figure 2.5. The recoding table shows the transfer digits t_j and the sum digits s_j for possible values of the input digit. The residual in this implicit recurrence is s_j.

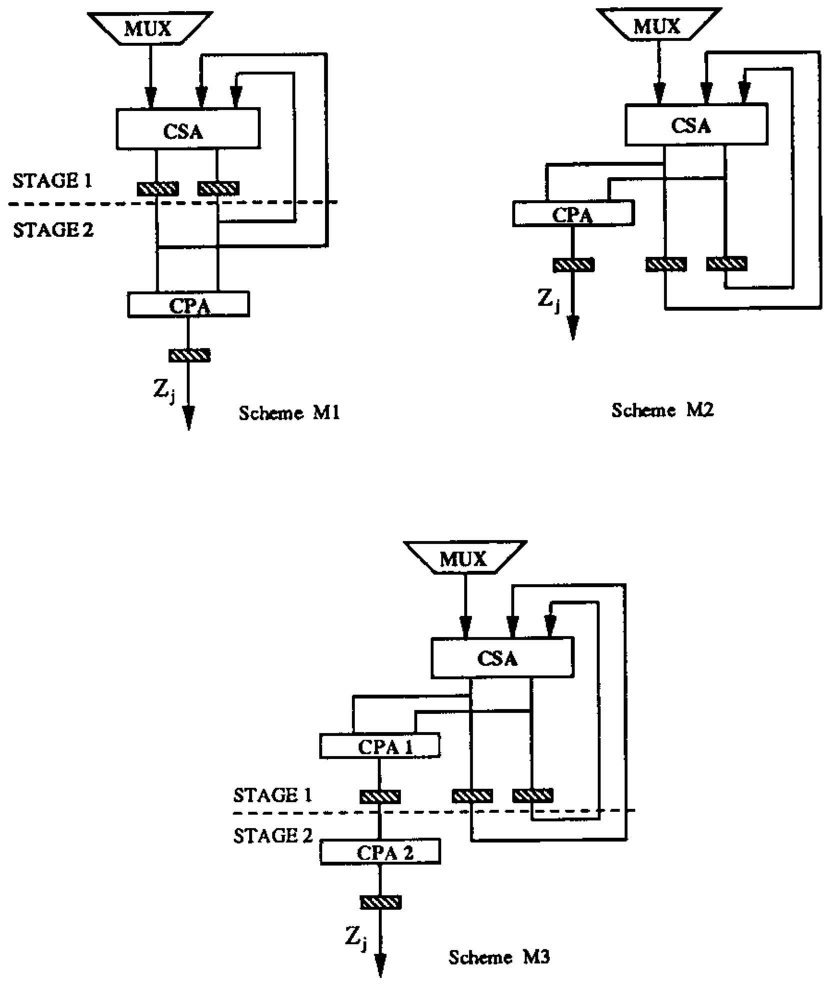


Figure 2.4: Multiplier Implementation Schemes

2. An explicit recurrence that produces output digits in the desired range without intermediate digits.

The design of a recoder using an explicit recurrence is described next.

The recoder is specified by:

- The radix r.
- The range of the over-redundant input integer Z_j (2's complement), $\underline{z} \leq Z_j \leq \overline{z}$.
- The digit set, $\{-\rho, \ldots, \rho\}$, of the output digit Y_j.

- The minimum precision, N, in digits, required for the recurrence. Consequently, the recoding delay is $N-1$ clocks. The value of N depends on the other specifications.

The recurrence for the recoder is given by

$$v[j+1] \;=\; r(v[j] - Y_j r^{N-1}) + Z_j, \quad v[0] = 0, \quad j = 0, 1, 2, \ldots \tag{2.8}$$

All values in the recurrence are integers. A recoder for $r = 4$ and $N = 3$ is shown as part of a module in Figure 2.9. It illustrates the recurrence step and the selection of the output digit, Y_j, based on the residual $v[j]$. In the j^{th} iteration, the over-redundant value Z_j is added to $v[j] - Y_j r^{N-1}$ to form $v[j+1]$.

An expression for the delay is obtained by considering the bounds on the shifted residual. Let the range of the shifted residual be defined by: $\underline{\xi} \leq v[j] - Y_j r^{N-1} \leq \overline{\xi}$. Since $v[j+1]$ must be contained in N digits for all values of residuals and all values Z_j, consider the case of the maximum residual and maximum Z_j. This is expressed by (2.9). Similarly, expression (2.10) considers the case of the minimum residual and minimum Z_j.

$$\overline{z} + r\overline{\xi} \;\leq\; \rho r^{N-1} + \overline{\xi} \tag{2.9}$$

$$\underline{z} + r\underline{\xi} \;\geq\; -\rho r^{N-1} + \underline{\xi} \tag{2.10}$$

Now consider the selection function to obtain expressions for $\overline{\xi}$ and $\underline{\xi}$. For a rounding selection function $\overline{\xi}$ and $\underline{\xi}$ are given by (2.11) and (2.12). A rounding selection function selects an output digit by *rounding* the residual to the nearest integer.

$$\overline{\xi} \;=\; \frac{1}{2} r^{N-1} - 1 \tag{2.11}$$

$$\underline{\xi} \;=\; -\frac{1}{2} r^{N-1} \tag{2.12}$$

Two values of N are obtained: Equations (2.9) and (2.11) give one, and Equations (2.10) and (2.12) give the other. The larger value of N satisfies both bounds and is expressed by:

$$N \;=\; 1 + \left\lceil \log_r \frac{2\max(\overline{z} + 1 - r, -\underline{z})}{2\rho + 1 - r} \right\rceil \tag{2.13}$$

The recoding delay, $N-1$, is defined as the number of clocks between *a* input digit and the output digit of *identical weight.* Given the radix, the input digit range, and the output digit set, the precision of the residual can be calculated to complete the design of the recoder.

2.3 Separate Multiplier (SM) Scheme

Figure 2.1 shows, the SM scheme to evaluate (1.1). The multiplier was discussed in Section 2.1. This section describes the recoder, the signed-digit adder and the performance of the scheme. Performance is measured by the sampling rate, given by the *clock period* times the *number of clocks* required for the critical loop. The critical loop consists of the multiplication $a \cdot y(n-1)$, the recoding of the output of the multiplication and the addition of R_j^{b+c}. The *clock period* is determined by the multiplier (Section 2.1). The *number of clocks* is determined primarily by the recoder and signed-digit adder. Given a digit set, the magnitude of the range of Z_j affects delays of the recoder and the adder, as described next.

Recoding Multiplier Output Z_j

The recoder converts the over-redundant integer Z_j into a smaller digit set acceptable to the input of the adder. The recoder described here uses a recoding table.

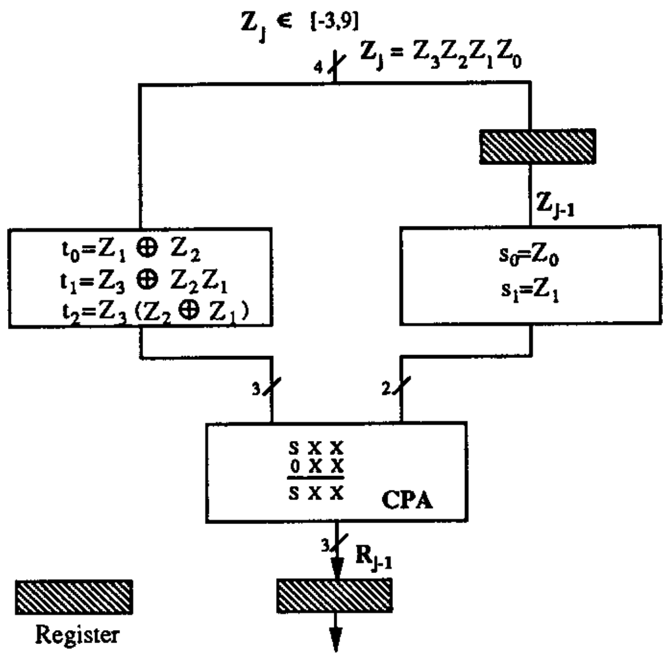
Figure 2.5 shows the recoding table for [-3,9] to [-3,3]. The table is implemented by a simple combinational network specified in Figure 2.5. Each digit, Z_j, input to the recoder, is converted to a transfer digit t_{j-1} and a sum digit s_j. The recoded digit R_{j-1} is produced by adding the current transfer digit t_{j-1} to the previous sum digit s_{j-1}.

The transfer digit t_{j-1} depends only on the value of Z_j. Thus, the on-line delay for recoding, called the recoding delay, is one clock. Since the recoded value must be registered after recoding (Figure 2.5), the total delay caused by inserting the recoder is two clocks.

As Table 2.1 shows, for a coefficient $|x| < 1$, the digit set [-3,3] has the minimum recoding delay. For $|x| < 2$, radices 4 and 8 cause less recoding delay than radix-2. The choice of radix-4 results in faster recoding, since radix-8 takes a longer clock period. Figure 2.6 shows the scheme for recoding from a digit set of [-9,9] to a digit set of [-3,3]. The recoding delay is three clocks. The signed-digit adder is considered next.

Signed-Digit Adder

Figure 2.7 shows two schemes for adding the recoded digits. The recoder outputs R_j^a, R_j^b, and R_j^c, are added and recoded again to produce Y_j in the appropriate digit set. The choice of radix-4 leads to fewer clocks for the addition. Equation

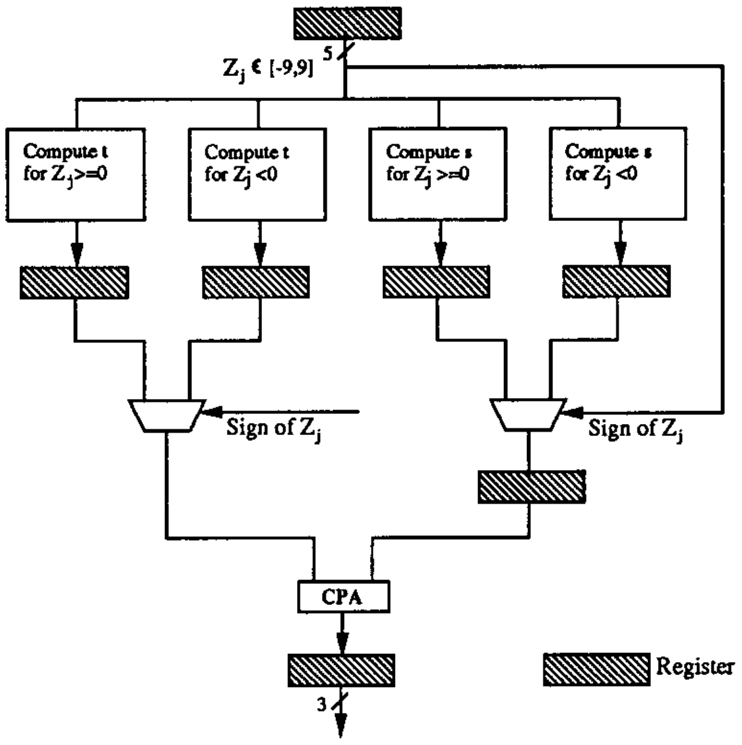


Z_j	-3	-2	-1	0	1	2	3	4	5	6	7	8	9
t_{j-1}	-1	0	0	0	0	1	1	1	1	2	2	2	2
s_j	1	-2	-1	0	1	-2	-1	0	1	-2	-1	0	1

Figure 2.5: Recoder for [-3,9] to [-3,3]

(2.13) indicates that a digit set of [-2,2] incurs a larger recoding delay in the adder than a digit set of [-3,3]. Thus a digit set of [-3,3] is chosen to reduce the delay for the entire module.

Scheme A1, shown in Figure 2.7, uses two adders, each adding two digits. For a digit set of [-3,3] both adders are identical, producing a digit in range [-3,3] (i.e., recoding from [-6,6] to [-3,3]) with a recoding delay of 1 clock. The total delay is four clocks as shown in Figure 2.7.



	Z_j	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9
$Z_{j+1} \geqslant 0$	t_{j-1}	-2	-2	-2	-1	-1	-1	-1	0	0	0	0	1	1	1	1	2	2	2	2
	S_j	-1	0	1	-2	-1	0	1	-2	-1	0	1	-2	-1	0	1	-2	-1	0	1
$Z_{j+1} < 0$	t_{j-1}	-2	-2	-2	-2	-1	-1	-1	-1	0	0	0	0	1	1	1	1	2	2	2
	S_j	-1	0	1	2	-1	0	1	2	-1	0	1	2	-1	0	1	2	-1	0	1

Figure 2.6: Recoder for [-9,9] to [-3,3]

Table 2.1: Variation of Multiplier Output Digit Z_j

Z_j	*Radix*	*Digit Set*	$\lvert$*Coefficient*$\rvert$	*Recoding Delay*
$[-1,3]$	2	$[-1,1]$	<1	2
$[-2,4]$	2	$[-1,1]$	<2	3
$[-2,8]$	4	$[-2,2]$	<1	2
$[-4,10]$	4	$[-2,2]$	<2	2
$[-3,9]$	4	$[-3,3]$	<1	1
$[-6,12]$	4	$[-3,3]$	<2	2
$[-4,18]$	8	$[-4,4]$	<1	2
$[-8,22]$	8	$[-4,4]$	<2	2

Scheme A2 adds all three digits in one stage, then recodes in the second and third stages. Scheme A2 requires a larger clock period due to the larger adder. Figure 2.6 shows the table for recoding from a digit set of [-9,9] to a digit set of [-3,3].

Performance

Figure 2.8 shows the block diagram of the radix-4 filter with digits in the set [-3,3] and coefficients with absolute value less than 2. The total delay is 9 clocks.

The estimated clock period for an implementation in a LCA10000 Array [LSI87] is 10ns. The maximum rate for this computation is determined by the delay of the critical loop. The critical loop consists of the path from the input $y(n-1)$ to the output $y(n)$. The critical loop delay is 7 clocks, for coefficients with magnitude < 2, because the two registers before the final adder can be eliminated. Thus, the estimated maximum rate is 14.3 Msamples/sec. The relatively low performance of this design is caused by excessive recoding delays. The next section describes a scheme which reduces the recoding delay.

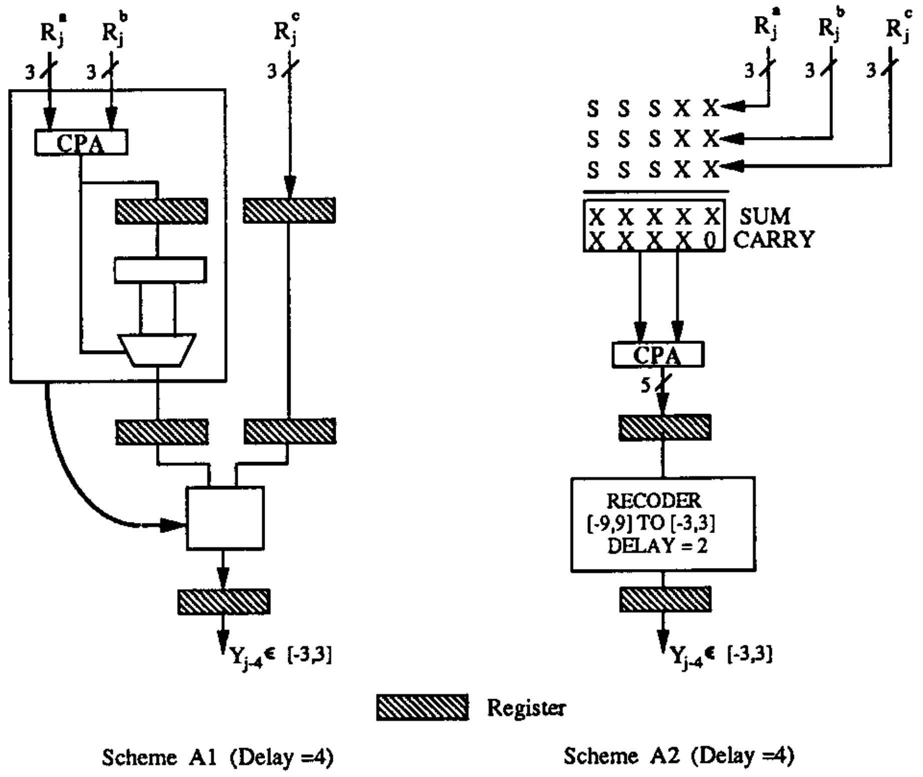


Figure 2.7: Signed-Digit Adder Implementation Schemes

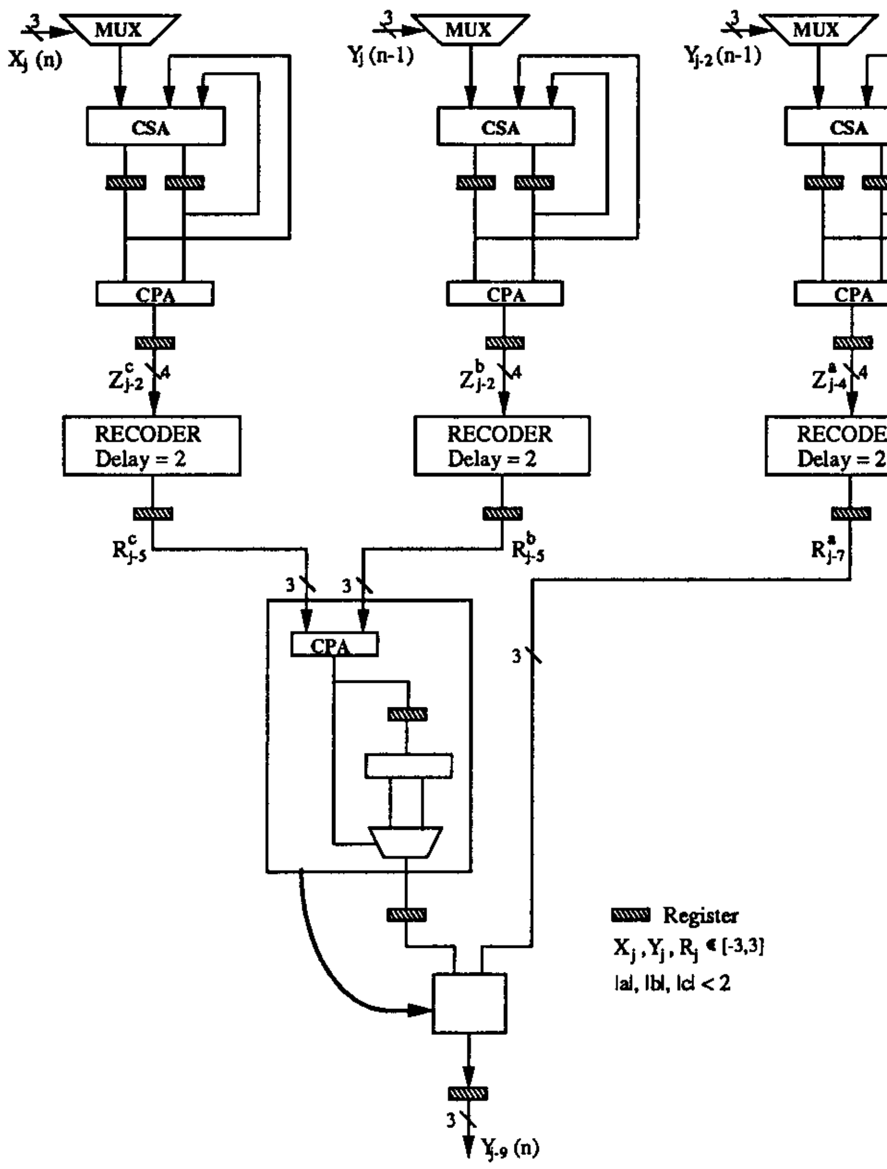


Figure 2.8: Separate Multiplier (SM) Scheme ($\delta_{imp} = 7$)

2.4 Combined Multiplier (CM) Scheme

The implementation outlined in Figure 2.8 shows that the on-line addition of the three signed digits R_j^a, R_j^b and R_j^c, takes 4 clocks. This section describes an implementation which performs the multiplication and addition in one carry-save adder array. The scheme has a delay of 6 clocks and requires fewer registers than the SM scheme. The combined multiply-add recurrence is derived by combining the three separate multiplication recurrences. The over-redundant digit produced by the multiply-add is recoded to obtain the output digit.

The Combined Multiply-Add Recurrence

The scheme shown in Figure 2.8 has three separate multipliers based on the recurrence defined by Equations (2.3) through (2.5). The three multiplication recurrences are given below.

$$\begin{aligned}
w^a[0] &= 0 \\
w^a[j+1] &= 4fraction(w^a[j] + aY_j(n-1)), \;\; j = 0, \ldots, d-1 \\
Z_j^a &= integer(w^a[j] + aY_j(n-1))
\end{aligned}$$

$$\begin{aligned}
w^b[0] &= 0 \\
w^b[j+1] &= 4fraction(w^b[j] + bY_j(n-2)), \;\; j = 0, \ldots, d-1 \\
Z_j^b &= integer(w^b[j] + bY_j(n-2))
\end{aligned}$$

$$\begin{aligned}
w^c[0] &= 0 \\
w^c[j+1] &= 4fraction(w^c[j] + cX_j(n)), \;\; j = 0, \ldots, d-1 \\
Z_j^c &= integer(w^c[j] + cX_j(n))
\end{aligned}$$

The three recurrences shown above are combined to produce a single recurrence:

$$w[0] = 0 \tag{2.14}$$

$$\begin{aligned}
w[j+1] = {} & 4fraction(w[j] + aY_j(n-1) + \\
& bY_j(n-2) + cX_j(n)), \;\; j = 0, \ldots, d-1
\end{aligned} \tag{2.15}$$

$$\begin{aligned}
Z_j = {} & integer(w[j] + aY_j(n-1)) + \\
& bY_j(n-2) + cX_j(n))
\end{aligned} \tag{2.16}$$

This recurrence is implemented by maintaining the partial product $w[j]$ in carry-save form, shown below, to allow the use of a carry-save adder.

$$w[j] = S[j] + C[j] \tag{2.17}$$
$$w[j] \geq 0 \tag{2.18}$$

Figure 2.9 shows the combined multiplier and the recoder. The range of the output digit of the combined multiplier, Z_j, is greater than for the case where the multipliers were separate. Table 2.2 shows the range of values of Z_j, and the resulting recoding delays, for different ranges of coefficients. The coefficients' ranges are expressed by s, defined as the sum of the maximum magnitudes of the coefficients. For example, $s = 4$ for $|a| < 2$, $|b| < 1$ and $|c| < 1$, and $s = 5$ for $|a| < 2$, $|b| < 1$ and $|c| < 2$. The delay of the recoder is obtained from expression (2.13). Table 2.2 shows that the minimum recoding delay is obtained for radix-4 with digit set [-3,3].

Table 2.2: Variation of Combined Multiplier Output Z_j

Radix	*Digit Set*	s	Z_j	*Recoding Delay*
2	[-1,1]	4	[-4,6]	4
2	[-1,1]	5	[-5,7]	4
4	[-2,2]	4	[-8,14]	3
4	[-2,2]	5	[-10,16]	3
4	[-3,3]	4	[-12,18]	2
4	[-3,3]	5	[-15,21]	2

Performance

The total delay is 6 clocks for $s = 4$ and $s = 5$. The clock period is determined by the first stage. For a LCA10000 Array [LSI87] the first stage takes 14ns. The CM scheme shown in Figure 2.9 may be pipelined at the first stage, i.e., the multiples $c \cdot X_j(n)$ and $b \cdot Y_j(n-2)$ are generated in the first stage. The second stage generates $a \cdot Y_j(n-1)$ and reduces 5 rows to 2. The clock period is reduced to 11ns and the maximum rate remains at 1 sample every 6 clocks, a sampling rate of 15Msamples/second.

The maximum rate of the CM scheme is about the same as that of the SM scheme. However, the CM scheme has a lower cost due to the fewer registers required for sum and carry.

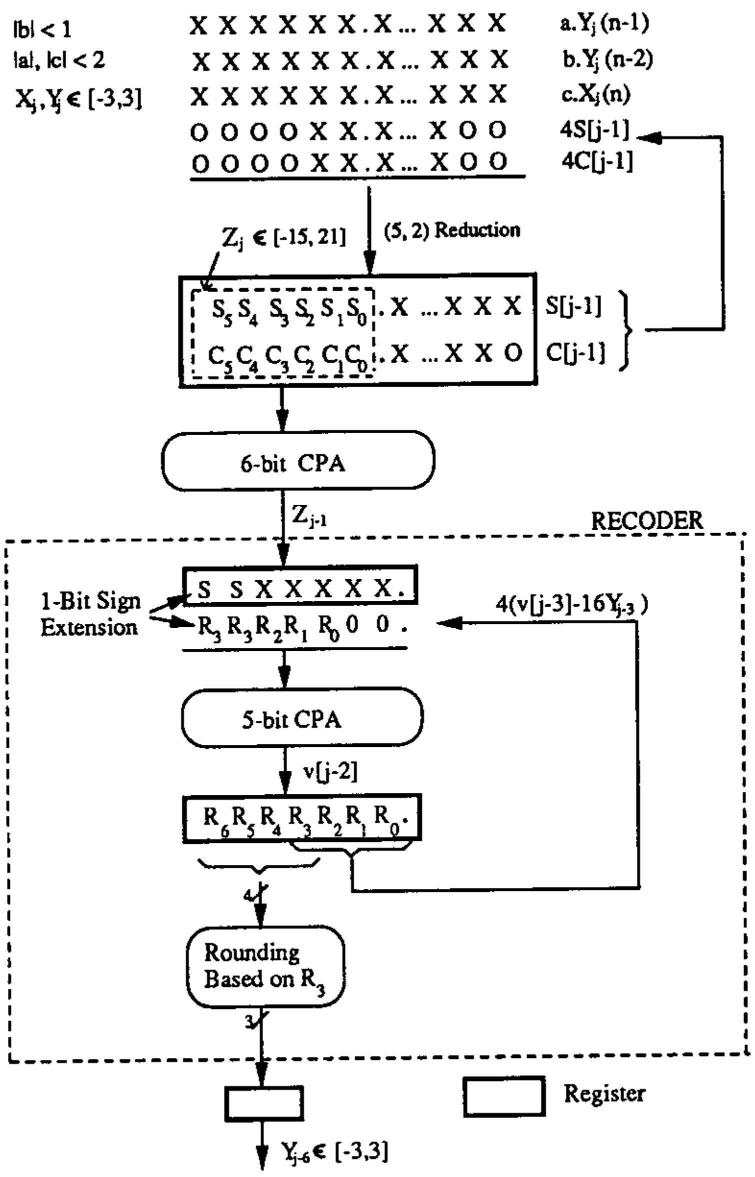

|b| < 1
|a|, |c| < 2
$X_j, Y_j \in [-3,3]$
X X X X X X.X... X X X a.Y_j(n-1)
X X X X X X.X... X X X b.Y_j(n-2)
X X X X X X.X... X X X c.X_j(n)
0 0 0 0 X X.X... X 0 0 4S[j-1]
0 0 0 0 X X.X... X 0 0 4C[j-1]
$Z_j \in [-15,21]$
(5, 2) Reduction
$S_5 S_4 S_3 S_2 S_1 S_0$.X ...X X X S[j-1]
$C_5 C_4 C_3 C_2 C_1 C_0$.X ...X X 0 C[j-1]
6-bit CPA
Z_{j-1}
RECODER
1-Bit Sign Extension
S S X X X X X.
$R_3 R_3 R_2 R_1 R_0$ 0 0 .
4(v[j-3]-16Y_{j-3})
5-bit CPA
v[j-2]
$R_6 R_5 R_4 R_3 R_2 R_1 R_0$.
4
Rounding Based on R_3
3
Register
$Y_{j-6} \in [-3,3]$

Figure 2.9: Combined Multiplier Scheme ($\delta_{imp} = 6$)

CHAPTER 3

A Composite On-Line Scheme

The schemes analyzed in Chapter 2 were composed of separate on-line units. The CM scheme described in Section 2.4 is an example. The scheme has long on-line delays caused by recoding stages. This chapter begins with a development of a composite on-line scheme based on a digit recurrence, similar to [Bra89a]. The entire computation expressed by (1.1) is carried out as a single operation in the composite on-line scheme.

An expression for the on-line delay of the composite scheme is derived for a N-order linear recurrence in Section 3.1. Section 3.3 describes how the parameters and architecture for a fast implementation of a second-order recurrence are selected. Performance and cost comparisons for four second-order radix-4 implementations are presented in Section 3.4.

3.1 Specification and Derivation of Recurrence

The notation used to specify a linear recurrence is introduced first. Expressions for input and output are derived in terms of the input digits, the on-line delay and the output digits. These expressions are used in the derivation of the digit recurrence. The notation adopted is as follows.

- Output expression for a linear N-order recurrence is given by
  $y(n) = cx(n) + a_1y(n-1) + a_2y(n-2) + \ldots + a_Ny(n-N)$
- Radix, r.
- Digit set $\{-\rho, \ldots, \rho\}$, for input, X_j, and output, Y_j.
- Number of radix-r digits per word, d.
- On-line delay for the recurrence, δ.
- Sum of the maximum absolute values of coefficients a_i and c denoted by s.

Definition 1 (δ) *The on-line delay, δ, of a digit recurrence algorithm is the delay measured in recurrence steps (or clocks), between* an *input digit and the output digit of* identical *weight.*

Definition 2 (s) *The sum of the bounds of the magnitude of coefficients in a digit recurrence is defined as:*

$$s = \sum_{i=1}^{N} A_i + C, \quad |a_i| < A_i, \quad |c| < C \tag{3.1}$$

The bounds on coefficient magnitudes are defined by stability criteria for recursive filters [PrMn88]. For a stable linear time-invariant system, the output, $y(n)$, must be bounded for every bounded input $x(n)$. Without loss of generality, $x(n)$ is assumed to be < 1. The stability of (1.1) and (3.1) is assured for coefficients inside the triangle shown in Figure 7.1. For a composite module implementing (1.1), with $|a| < 2$ and $|b| < 1$ for stability, $s = C+3$. To derive a digit recurrence for the composite on-line scheme, the ouput and input values must be expressed in terms of their digits and the on-line delay. Consider first the output of the on-line computation. The output contains δ leading zeros. Expressed as a full-precision scaled result, the output is given by:

$$y(n) = r^{-\delta}(cx(n) + \sum_{k=1}^{N} a_k y(n-k)) \tag{3.2}$$

The input values $x(n)$ and $y(n-k)$ are expressed in terms of their digits as follows.

$$x(n) = \sum_{j=0}^{d+\delta-1} X_j(n) r^{-j}, \quad X_j = 0 \text{ for } j \geq d \tag{3.3}$$

$$y(n-k) = \sum_{j=0}^{d+\delta-1} Y_j(n-k) r^{-j}, \quad Y_j(n-k) = 0 \text{ for } j \geq d \tag{3.4}$$

Note that $d+\delta$ digits of $x(n)$ and $y(n-k)$ are required to produce δ zeros and d digits of $y(n)$. Consequently δ zero digits have to be appended after the least significant input digit. At the end of the j^{th} iteration, $j+1$ of the most significant digits of each input term have been consumed. The values of the inputs, represented by the $j+1$ most significant digits, are given by:

$$x(n)[j] = x(n)[j-1] + r^{-j} X_j(n), \quad x(n)[-1] = 0 \tag{3.5}$$

$$y(n-k)[j] = y(n-k)[j-1] + r^{-j} Y_j(n-k), \quad y[-1] = 0 \tag{3.6}$$

Deriving a Recurrence

At the end of each iteration, one output digit Y_j is produced based on the value of the scaled residual $w[j]$. The scaled residual at the end of the j^{th} iteration is defined as the difference between the scaled actual sum (obtained by using the $j+1$ most significant digits of the inputs) and the scaled computed sum (based on the j output digits produced thus far). The scaled residual is expressed as follows.

$$\begin{aligned} w[-1] &= 0 & (3.7) \\ w[j] &= r^j\{r^{-\delta}(cx(n)[j] + \sum_{k=1}^{N} a_k y(n-k)[j]) \\ &\quad -y(n)[j-1]\}, \;\; j = 0,1,2\ldots & (3.8) \end{aligned}$$

Using Equations (3.5), (3.6) and (3.8), a recurrence in terms of the residual $w[j]$ is formed.

$$\begin{aligned} w[-1] &= 0 & (3.9) \\ w[j] &= r\{w[j-1] - Y_{j-1}(n)\} + \\ &\quad r^{-\delta}\{cX_j(n) + \sum_{k=1}^{N} a_k Y_j(n-k)\}, \;\; j = 0,1,2\ldots & (3.10) \end{aligned}$$

Digit Selection and Residual Bounds

The output digit Y_j is selected based on the value of $w[j]$, i.e., $Y_j = sel(w[j])$. The selection function maps $w[j]$ into the digit set $\{-\rho,\ldots,\rho\}$ such that the residual is bounded as follows.

$$\underline{\xi} \leq (w[j] - Y_j(n)) \leq \overline{\xi} \qquad (3.11)$$

The values of $\overline{\xi}$ and $\underline{\xi}$ depend on the representation of the residual and the selection function. In general $\overline{\xi} \neq -\underline{\xi}$. For a rounding selection function with a non-redundant representation of the residual, $\overline{\xi} = -\underline{\xi}$.

3.2 Expression for On-Line Delay

The on-line delay is obtained from (3.10) by calculating the maximum and minimum values of $w[j]$, which guarantee a bounded residual. The maximum value of $w[j]$ occurs when the maximum residual is added to the maximum values of the coefficients' multiples. The minimum value of $w[j]$ occurs when the minimum residual is added to the minimum values of the coefficients' multiples. The maximum and minimum value cases may be expressed as follows, using the bounds specified in (3.11) and the definition of s in (3.1).

$$\overline{\xi} + \rho = r\overline{\xi} + s\rho r^{-\delta} \qquad (3.12)$$
$$\underline{\xi} - \rho = r\underline{\xi} - s\rho r^{-\delta} \qquad (3.13)$$

To obtain an expression for δ, $\overline{\xi}$ and $\underline{\xi}$ must be evaluated by considering the residual and the selection function. The residual $w[j]$ is maintained in carry-save form for a fast recurrence step. To reduce $\overline{\xi}$ and $-\underline{\xi}$, and perform digit selection, the integer part and β fraction bits of the sum and carry words (representing $w[j]$) are added by a CPA. Thus $w[j]$ is represented in the partial carry-save form shown below.

$$\begin{array}{c} O\ O\ O\ .\overbrace{O\ O\ \ldots\ O}^{\beta}\ X\ X\ \ldots\ X \\ X\ X\ X\ .X\ X\ \ldots\ X\ X\ X\ \ldots\ X \end{array}$$

If rounding selection is based on this partial carry-save representation of $w[j]$, then $\overline{\xi}$ and $\underline{\xi}$ are given by (3.14) and (3.15).

$$\overline{\xi} = 0.5 + 2^{-\beta} + \varepsilon \qquad (3.14)$$
$$\underline{\xi} = -0.5 \qquad (3.15)$$

The small value ε depends on the precision of the partial carry-save form. If f is the number of fraction bits in the partial carry-save form, $\varepsilon = -2^{1-f}$.

Consider the maximum values of the two sides of (3.10):

$$\begin{aligned} \max(w[j]) \leq\ & \max(r\{w[j-1] - Y_{j-1}(n)\}) + \\ & \max(r^{-\delta}\{cX_j(n) + \sum_{k=1}^{N} a_k Y_j(n-k)\}) \end{aligned} \qquad (3.16)$$

The maximum values in (3.16) can be substituted using (3.11) and (3.14) to get:

$$0.5 + 2^{-\beta} + \rho + \varepsilon \leq r(0.5 + 2^{-\beta} + \varepsilon) + r^{-\delta} s\rho \qquad (3.17)$$

Solving 3.14 yields an expression for δ:

$$\delta = \left\lceil \log_r \frac{s\rho}{\rho - (r-1)(0.5 + 2^{-\beta} + \varepsilon)} \right\rceil \qquad (3.18)$$

A slightly different expression for δ can be derived by considering the minimum values of (3.10) and Equations (3.11) and (3.15). This value of δ may not satisfy the maximum bound on the residual.

Thus δ may be calculated using Equation (3.18) given the digit set, the radix, the filter expression and β. For a second-order recurrence, Table 3.1 shows the on-line delays for different maximum magnitudes of coefficients, radices and digit sets. The minimum value of δ is obtained when β is infinite, which requires all fraction bits to be included in the CPA. A higher value of δ may result if β is small. Thus β is chosen to minimize the clock period, by making a suitable compromise between CPA length and δ. For a given digit set and radix, the on-line delay is decremented once every time s is scaled down by the radix. This trade-off between coefficient magnitudes and δ is seen in (3.18).

Table 3.1: Variation of On-Line Delay

s	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6
Radix		2			4			4			8			8			8	
ρ		1			2			3			4			5			6	
β	2	3	3	3	4	4	2	3	3	5	6	6	3	3	3	2	2	2
δ		4			3			2			2			2			2	

3.3 Second-Order Radix-4 Implementations

This section outlines four implementations based on design parameters selected from Table 3.1. The problem of selecting the architecture that yields the fastest rate is discussed first. Circuit-level improvements that minimize critical path delay and reduce the number of gates are described later. The on-line delay of an implementation, δ_{imp}, is defined next.

Definition 3 (δ_{imp}) *For a digit recurrence implementation, δ_{imp} is the minimum number of clocks between an input digit and the output digit of* identical *weight. For any digit recurrence implementation, $\delta_{imp} \geq 1 + \delta$.*

For the architectures described here, $\delta_{imp} = \delta + 2$. The 2 added to δ corresponds to the number of stages: one for implementing the digit recurrence and one for latching the selected output digit.

Choosing a Maximum Rate Architecture

A maximum rate architecture minimizes $\delta_{imp}t_{clk}$. Since δ_{imp} is the sum of δ and the number of stages, digit recurrence parameters must be chosen to minimize δ and t_{clk}. In addition, the architecture must be designed to minimize t_{clk} and the number of stages. The choice of parameters is discussed first and the architecture is discussed next.

The parameters of the recurrence are expressed in (3.18). Given s, the task is to determine δ, β, r and ρ such that t_{clk} is minimized. The value of t_{clk} increases with r and ρ. Large values of β also increase t_{clk}. Table 3.1 shows the design space for small r. The choice between radix-4 and radix-8 is clear, since δ is 2 and radix-4 results in lower t_{clk}. The choice between radix-2 and radix-4 depends on the technology. As discussed in Section 2.1, the radix-4 implementation is about 13% slower than the radix-2 implementation. However, the radix-2 scheme has a larger δ which more than offsets the faster clock rate. Thus the radix-4 design with $\rho = 3$ is the best choice for maximum rate in LCA10000 Array technology [LSI87]. The choice of $\rho = 3$ requires precomputing and storing the coefficient multiples of 3. To quantify the maximum rate differences between $\rho = 3$ and $\rho = 2$, designs for both digit sets are implemented.

Choosing the architecture consists mainly of determining the number of stages in the pipeline. Since the digit recurrence requires feedback, the number of stages in the pipeline cannot be increased arbitrarily. The number of stages that minimize $\delta_{imp}t_{clk}$ is determined by a delay analysis. Consider a 2-stage design. The first stage implements multiplication and addition to compute $w[j]$, and the second stage implements digit selection. The first stage takes 14ns and the second 6ns. Thus $t_{clk} = 14$, and the total delay is 56ns. For a single stage design, $t_{clk} \approx 20ns$ (i.e., 14+6), and $\delta_{imp} = 3$, giving a total delay of 60ns. Thus a 2-stage design is faster than a single stage one. The first stage can be pipelined further to obtain a faster clock. This is discussed later in the chapter.

For designs with different number of stages with the same total delay, the cost-effective choice is the design with the faster clock. For example, consider a 2-stage design with $t_{clk} = 15$ and $\delta_{imp} = 4$, and a 3-stage design with $t_{clk} = 12$ and $\delta_{imp} = 5$. Both have a total delay of 60ns. The number of modules required to achieve the maximum rate is given by $\lceil \frac{d+\delta}{\delta_{imp}} \rceil$. With $\delta = 2$ and $d = 8$, the number of modules required for maximum rate is 3 for the 2-stage design, and 2 for the 3-stage design. The following sections consider 2-stage and 3-stage designs.

Circuit-Level Optimizations

Figure 3.1 shows a 2-stage design, for $\rho = 3$ and $s = 6$, and Figure 3.2 shows two optimizations that reduce the number of gates and minimize potential critical path delay. The first optimization includes the rounding selection in the CSA array. The second optimization reduces the number of gates by optimizing the sign extension required by 2's complement. The rounding optimization is discussed next.

As Equation (3.10) shows, the selected digit Y_{j-1} must be subtracted from the residual $w[j-1]$ in each iteration. For a rounding selection function, the integer part produced by the CPA is incremented by the leading fraction bit to produce the selected digit. Figure 3.1 shows the selection step based on the four most significant bits produced by the CPA.

The selection may be optimized by eliminating the separate increment step and including the increment in the CSA array. The selection optimization, along with additional logic simplification, is shown in Figure 3.2. Adding $\frac{1}{2}$ to the residual produces the correct digit for rounding selection. To compensate for the addition, the scaled residual must be reduced by 2 in the next iteration. As shown in Figure 3.2, this is accomplished by adding -2 to the scaled residual (the $110.1X\ldots$ term). A side effect of this addition is that a -2 output digit is produced, instead of a 0 digit, immediately after the sum and carry registers are cleared. This is remedied by clearing the output digit register a clock later.

Figure 3.3 shows the optimized 2-stage scheme chosen for implementation. The critical path consists of the buffers which fan out the digit to the multiplexers, the 7-input multiplexes which select the coefficient multiple, and the 5-to-2 CSA tree. The critical path can be shortened by:

1. Selecting a small digit set, i.e., $\rho = 2$.
2. Introducing another stage for multiplexing the coefficients of digits $Y_j(n-2)$ and $X_j(n)$ as shown in Figure 3.4. This stage does not reduce the maximum rate because the delay incurred by $Y_j(n-1)$ is still the same.

Consequently, four designs are considered for implementation:

1. 2-stage design with $\rho = 3$ and $\delta_{imp} = 4$.
2. 3-stage design with $\rho = 3$ and $\delta_{imp} = 4$.
3. 2-stage design with $\rho = 2$ and $\delta_{imp} = 5$.

4. 3-stage design with $\rho = 2$ and $\delta_{imp} = 5$.

The designs are implemented in LSI Logic Corporation's LCA 10000 Series Compacted Array Technology [LSI87]. The implementations do not include the I/O buffers and pads. The next section compares the performance and cost of the implementations.

3.4 Performance and Cost

This section compares the performance (number of samples per second) and cost (number of gates), based on the simulation results obtained from the implementations. Comparisons are meaningful only when the following factors are identical for all implementations.

1. Technology.
2. Sum of coefficient bounds, s, defined in (3.1).
3. Number of digits, d, per word of input and output.
4. Number of bits, m, used in the implementation to represent coefficient in 2's complement.

Typically, m or d is specified and the number of fraction bits required in the implementation, f, must be determined. For the radix-4 implementations described, the three values are related as follows.

$$m = 2d \tag{3.19}$$
$$f = m + 2(\delta - 1) \tag{3.20}$$

The cost measures derived from the simulation are the number of gates used for the most significant part of the module and the number of gates per fraction bit slice. The most significant part includes the integer bits, a few fraction bits (typically β) and the selection logic. The number of gates used for a similar implementation with f fraction bits are given by the following expressions.

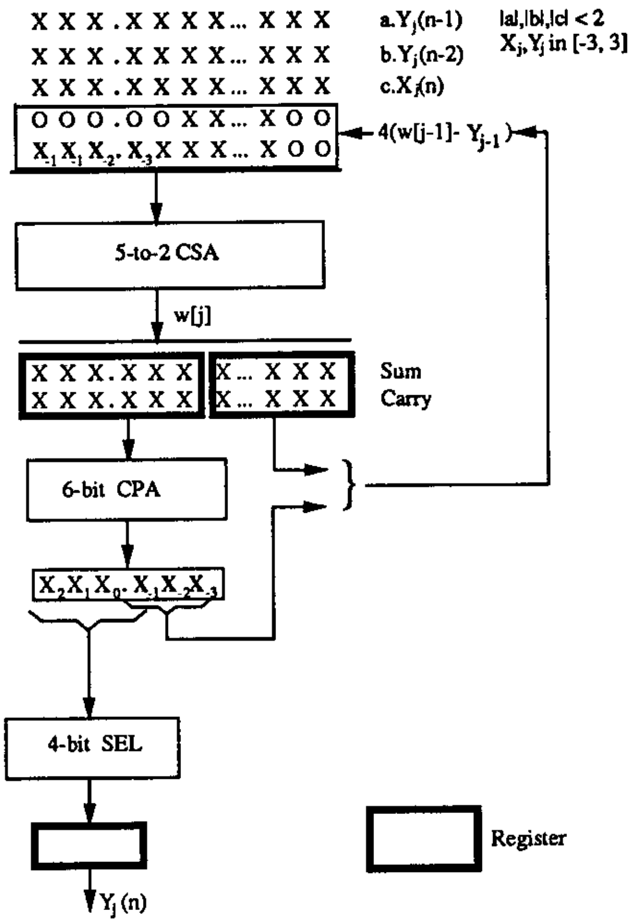


Figure 3.1: Composite 2-Stage On-Line Scheme ($\delta_{imp} = 4$)

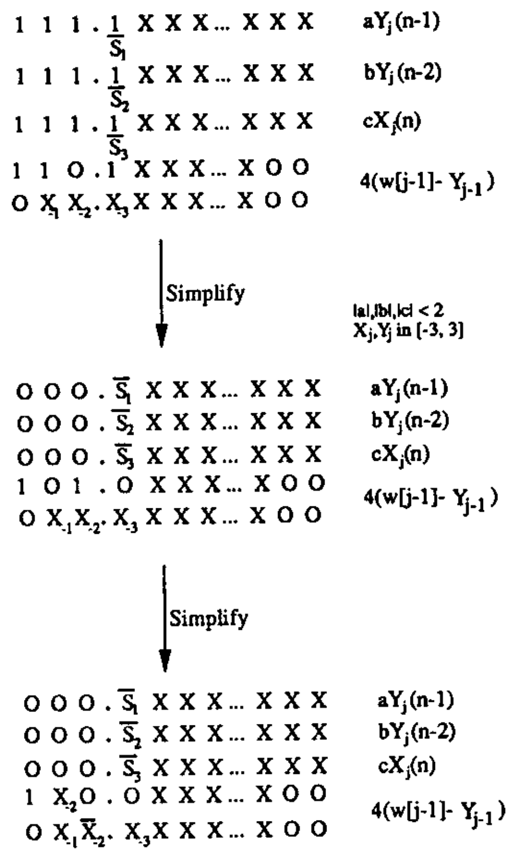


Figure 3.2: Optimizing Composite On-Line Scheme

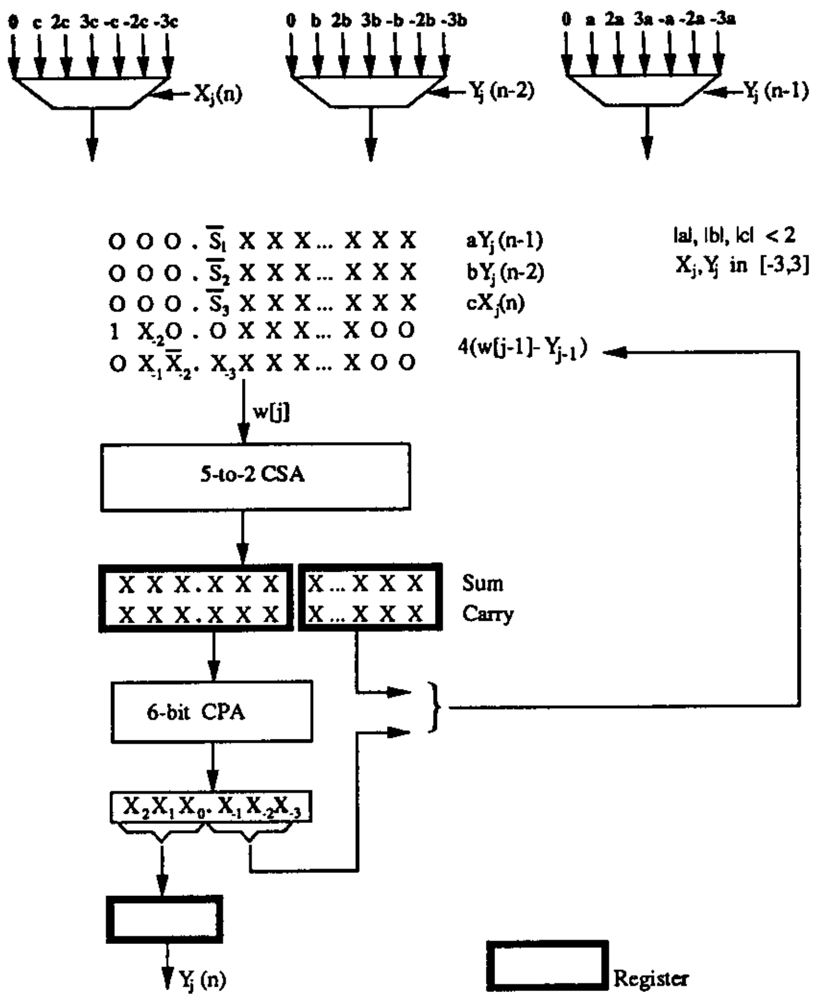


Figure 3.3: Optimized 2-Stage Composite On-Line Scheme ($\delta_{imp} = 4$)

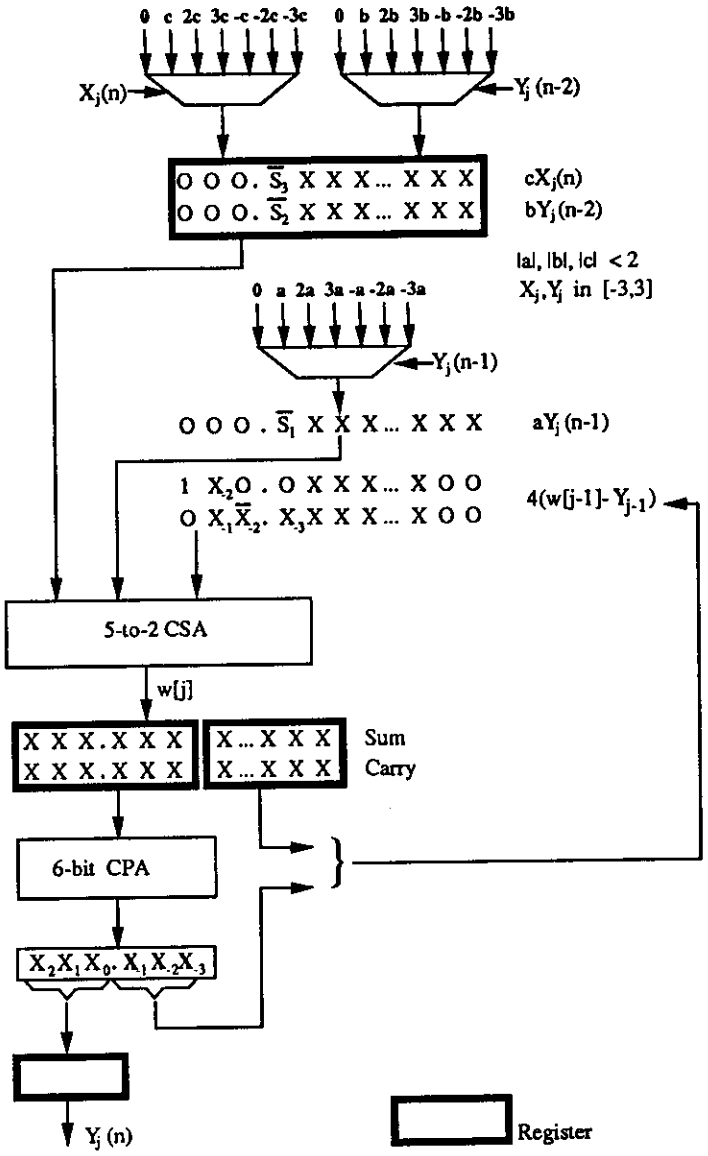


Figure 3.4: Optimized 3-Stage Composite On-Line Scheme ($\delta_{imp} = 4$)

Composite 2-stage ($\rho = 2$):	$\text{Gates}_{\text{module}} = 372 + 89f$
Composite 2-stage ($\rho = 3$):	$\text{Gates}_{\text{module}} = 336 + 125f$
Composite 3-stage ($\rho = 2$):	$\text{Gates}_{\text{module}} = 390 + 109f$
Composite 3-stage ($\rho = 3$):	$\text{Gates}_{\text{module}} = 354 + 145f$

Expressions for three performance measures are given below. The maximum rate and the rate of a single module are obtained by finding the minimum clock period, t_{clk} (ns), at which the implementations can be properly simulated. The minimum clock period increases very little with higher precision. The increase is purely due to the larger fan-out caused by the increased number of digit multiplexers.

1. Rate of a single module (Msample/s) $= \frac{1000}{t_{clk}(\delta+d)}$
2. Maximum Rate (Msamples/s) $= \frac{1000}{t_{clk}\delta_{imp}}$
3. Number of modules for Maximum Rate $= \left\lceil \frac{d+\delta}{\delta_{imp}} \right\rceil$

Table 3.2 shows the performance and cost for four implementations with I/O words of 16 bits ($d = 8$) and $s = 6$. The fastest design has 3 stages and uses a digit set of $\{-3, \ldots, 3\}$. The latency of the 3-stage modules from the $x(n)$ or the $y(n-2)$ input is 5 clocks. However, the maximum rate is determined by the latency from the $y(n-1)$ input, δ_{imp}, which is 4 clocks. The maximum rate of 1 sample per 4 clocks can be achieved using three modules. The number of *Gates/Array* is the total number of gates for as many modules as required to produce the maximum rate. Arrays are described in Chapter 5.

Table 3.2: Performance/Cost for Composite Word Modules ($d = 8$)

Module Type	ρ	δ_{imp}	*Gates /Mod.*	t_{clk} *ns*	*Max. Rate (MSmp./s)*	*Gates /Array*
Composite 2-stage	2	5	2152	14	14.3	5976
Composite 2-stage	3	4	2586	14	17.8	6798
Composite 3-stage	2	5	2570	11	18.8	7230
Composite 3-stage	3	4	2964	11	22.7	7932

CHAPTER 4

Schemes Using On-Line Multiply-Add Modules

The Multiply-Add (MA) module is a useful building block in realizing conventional (bit parallel) FIR and IIR filters. The *on-line* MA module has a similar usefulness in building *on-line* filters, IIR in particular [Bra89a]. Sections 4.1 and 4.2 describe the design and implementation of an on-line MA module that is less pipelined but much faster than the implementation described in [Bra89a]. The design choices and decisions are similar to those faced in implementing a composite on-line module described in the previous chapter. Section 4.3 discusses performance and costs of two MA modules. It also compares the performance and cost of MA module implementations that compute (1.1) with the composite module implementations described in the previous chapter.

4.1 Design of MA Module

The computation performed by the on-line MA module is generically expressed by $y = cx + u$, where x, y and u are on-line forms. Since the computation is a specific instance of Equation (1.1), with $b = 0$ and $a = 1$, the problem of designing a fast MA module is similar to that of designing a composite module which implements (1.1). The recurrence for the MA, an instance of (3.10), is given by:

$$\begin{aligned} w[-1] &= 0 & (4.1)\\ w[j] &= r\{w[j-1] - Y_{j-1}(n)\} + \\ &\quad r^{-\delta}\{cX_j(n) + U_j(n)\}, \;\; j = 0,1,2\ldots & (4.2) \end{aligned}$$

To achieve a fast implementation of the recurrence, the following parameters have to be selected.

- On-line delay δ and module delay δ_{imp}.
- Radix r and digit set $\{-\rho, \ldots, \rho\}$ for X, Y and U.
- Number of fraction bits required for selection, β.

The stability limit for second-order filters requires that $|c| < 2$. Consequently, $s = 3$ for an MA module. Equation (3.18) can be used to select appropriate parameters. Table 4.1 shows the variation of δ and β for radices 2, 4 and 8. The lowest on-line delay is obtained with a digit set of {-6, ..., 6} for a radix-8 implementation. The advantage of lower on-line delay is offset by the longer clock period required for the larger digit set. Similarly, the radix-2 design has a slightly shorter clock period which is more than offset by the larger on-line delay. This leaves the radix-4 designs for implementation. The implementations and circuit-level optimizations for the radix-4 designs are discussed next.

Table 4.1: Variation of On-Line Delay for MA Module ($s = 3$)

Radix	2	4	4	8	8	8	8
ρ	1	2	3	4	5	6	7
β	3	5	2	5	3	5	3
δ	3	2	2	2	2	1	1

4.2 Implementation of MA Module

The implementation of the radix-4 MA modules is similar to the 2-stage composite module shown in Figure 3.1. The differences are in the number of terms and the length of the CPA. Figure 4.1 shows the the circuit-level optimizations for the MA modules. Two of these optimizations, performing rounding selection in the CSA and logic minimization, were discussed in Section 3.3 for the composite module. The third optimization replaces a 4-2 CSA with a 3-2 CSA, as shown in Figure 4.1. This is accomplished by observing the following.

1. The $U_j(n)$ term requires only 4 fraction bits.
2. The shifted Partial Carry-Save form has $\beta - 2$ zeros in the most significant fraction part due to the CPA.
3. If β is taken to be 6, instead of 5 as shown in Table 4.1, then $U_j(n)$ can be fitted into the position occupied by the zeros in the partial carry-save form (Section 3.2).

Thus a 3-2 CSA can be used instead of a 4-2 CSA, reducing the critical path delay by as much as t_{fa}, if the longer CPA does not become the new critical path. For the radix-4 case with $\rho = 2$, the CPA is extended by only 1 bit.

Simulation shows that the CPA is not the critical path and the optimization increases the clock rate. Figure 4.2 shows the block diagram for a radix-4 ($\rho = 2$) optimized MA module. The performance and costs of MA modules and word modules using MA modules are discussed next.

4.3 Performance and Cost

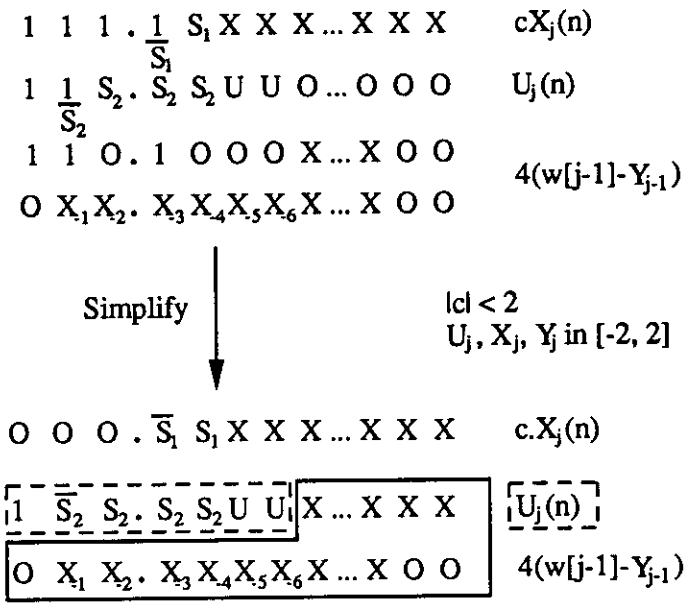
Much of the discussion on the performance and costs of composite modules apply to *word modules* composed of MA modules. A word module is a module that produces all digits of an output word at the same digit output. As mentioned earlier, the simulations provide the minimum clock period for each design. The cost figures obtained from the implementation are the total number of gates for each design and the number of gates for a bit-slice of the fraction part.

Table 4.2 summarizes the results for the two MA modules ($d = 8$), implemented in a LCA10000 Array [LSI87]. The table shows that the module with a digit set of {-2, ..., 2} achieves a higher clock rate using a smaller number of gates. The following expressions give the number of gates as a function of the number of fraction bits used in the implementation. Equations (3.19) and (3.20) apply.

$$\text{MA Module } (\rho = 2): \quad \text{Gates}_{\text{module}} = 373 + 45f$$

$$\text{MA Module } (\rho = 3): \quad \text{Gates}_{\text{module}} = 374 + 54f$$

A word module to realize Equation (1.1) can be implemented using three MA modules. Table 4.3 compares the performance and costs of word modules composed of MA modules with the composite modules described in Chapter 3. For all schemes shown in the table the coefficient precision, m, is 16 bits, and the I/O word length, d, is 8 digits. The gate counts for the arrays assumes one set of coefficient latches.



1 1 1 . 1 S1 X X X ... X X X
S̄1
cXj(n)
1 1 S2 . S2 S2 U U O ... O O O
S̄2
Uj(n)
1 1 O . 1 O O O X ... X O O
O X-1 X-2 . X-3 X-4 X-5 X-6 X ... X O O
4(w[j-1]-Yj-1)
Simplify
|c| < 2
Uj, Xj, Yj in [-2, 2]
O O O . S̄1 S1 X X X ... X X X
c.Xj(n)
1 S̄2 S2 . S2 S2 U U X ... X X X
Uj(n)
O X-1 X-2 . X-3 X-4 X-5 X-6 X ... X O O
4(w[j-1]-Yj-1)

Figure 4.1: Optimizing Multiply-Add (MA) Scheme

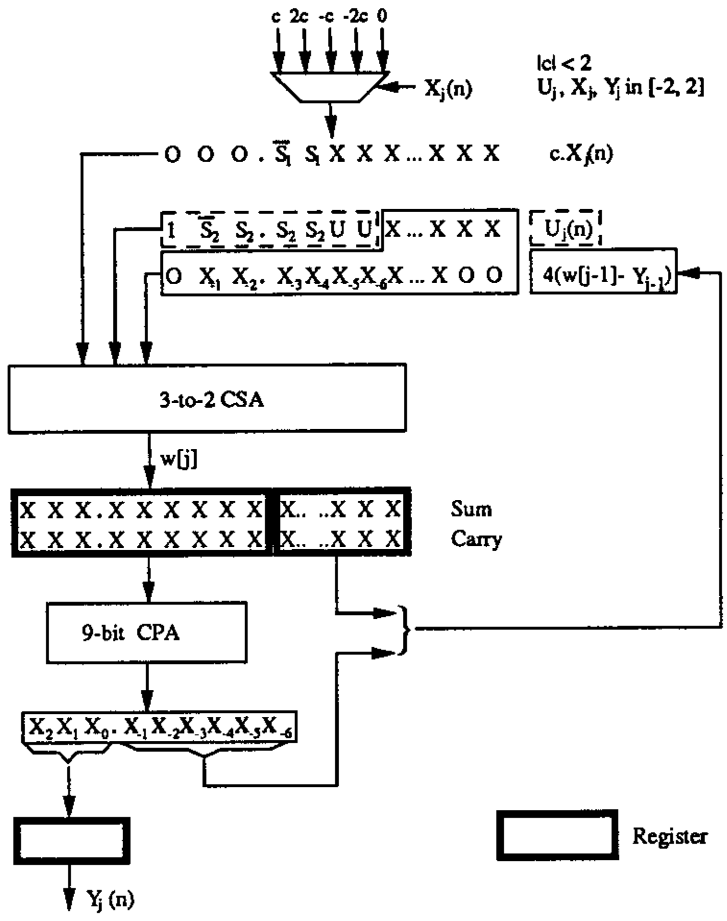

c 2c -c -2c 0
Xj(n)
|c| < 2
Uj, Xj, Yj in [-2, 2]
O O O . S1 S1 X X X ... X X X
c.Xj(n)
1 S2 S2 . S2 S2 U U X ... X X X
Uj(n)
O X1 X2 . X3 X4 X5 X6 X ... X O O
4(w[j-1] - Yj-1)
3-to-2 CSA
w[j]
X X X . X X X X X X X.. ..X X X
Sum
Carry
9-bit CPA
X2 X1 X0 . X-1 X-2 X-3 X-4 X-5 X-6
Register
Yj (n)

Figure 4.2: Optimized On-Line Multiply-Add (MA) Scheme ($\delta_{imp} = 4$)

Conclusions

The fastest design is the array of word modules composed of MA modules with $\rho = 2$. This array is the most modular design, the best choice for custom implementation. The most cost-effective design is the composite 3-stage module with $\rho = 3$. Among the composite modules, the 3-stage designs are faster and more cost-effective than the 2-stage designs. The smaller digit set does not yield higher maximum rates for composite modules due to the larger on-line delay. The maximum rate can be produced by an array of modules, as discussed in Chapter 5. Table 4.3 shows the number of modules required to produce the maximum rate as *Modules/Array*, and the total number of gates taken by these modules as *Gates/Array*.

Table 4.2: Performance/Cost for MA Modules ($d = 8$)

Module Type	ρ	δ_{imp}	*Gates per Module*	t_{clk} *(ns)*	*Gates per frac. bit-slice*
MA	2	4	1183	9	45
MA	3	4	1346	10	54

Table 4.3: Comparison of Performance/Cost of Word Modules ($d = 8$)

Module Type	ρ	δ_{imp}	*Gates per WM*	t_{clk} *(ns)*	*Max. Rate (MSmp./s)*	*Gates /Array*	*Modules /Array*
Comp. 2-stage	2	5	2152	14	14.3	5976	3
Comp. 2-stage	3	4	2586	14	17.8	6798	3
Comp. 3-stage	2	5	2570	11	18.8	7230	3
Comp. 3-stage	3	4	2964	11	22.7	7932	3
MA-based	2	4	3549	9	27.8	10167	9
MA-based	3	4	4038	10	25.0	11154	9

CHAPTER 5

Multi-Module Arrays

This chapter describes the structure, synchronization, performance, and cost of multi-module arrays composed of modules developed in previous chapters. The arrays compute the second-order recurrence specified by Equation (1.1). The sampling rate of the multi-module array is determined by the type of module composing the array. Four types of arrays are discussed:

1. Array of Composite Word Modules

2. Array of MA Word Modules

3. Array of Composite Digit Stages

4. Array of MA Digit Stages

Word Modules and Digit Stages

Arrays are classified as word module or digit stage arrays [ErLn89]. A word module has one digit-serial output that produces all digits of a specific word. All designs developed in previous chapters are word modules. One carry-save adder stage is used for all iterations. As Figure 3.3 shows, the current residual is fed back as a shifted residual for the next iteration.

A digit stage array is derived from a word module by executing each iteration in a separate stage called a digit stage. Each digit stage accepts specific input digits and produces a specific output digit. The residual is passed from one stage to the next. Consequently, the width of a stage is one digit less than that of the previous stage. Arrays of digit stages are discussed in Sections 5.3 and 5.4.

5.1 Array of Composite Word Modules

This section discusses the structure, synchronization, performance, and cost of arrays of composite word modules. Table 3.2 summarizes the performance and cost of composite word modules developed in Chapter 3. Arrays may be based on any of the modules in Table 3.2.

Figure 5.1 shows the structure of an array of composite word modules with $\delta_{imp} = 4$ and $d = 8$. The number of modules required for maximum rate is given by $\lceil \frac{\delta+d}{\delta_{imp}} \rceil$. The topology of the array is a ring, though drawn as a linear array for convenience. Three digit inputs accept $x(n)$, $x(n+1)$, and $x(n+2)$, and three digit outputs produce $y(n)$, $y(n+1)$, and $y(n+2)$.

Each module produces all digits of a word: module 1 produces $y(n)$, $y(n+4)$, $y(n+8)$ $\cdots$, module 2 produces $y(n+1)$, $y(n+5)$, $y(n+9)$ $\cdots$, and so on. The inputs to the array are shown relative to a time scale at the top of Figure 5.2. The y inputs are omitted from the figure for clarity. The inputs are skewed by δ_{imp} clocks. Input words flowing into a given module are separated by four zeros. If each module is fully utilized, input words are separated only by two zeros. Full utilization is achieved when $\delta + d$ is an integer multiple of δ_{imp}. Since $\delta_{imp} = 4$, four digit registers are required to synchronize the delayed y input (i.e., the $y(n-2)$ input in (1.1)).

An array of 3-stage composite word modules, with $\delta_{imp} = 4$, is almost identical to the array of 2-stage composite word modules shown in Figure 5.1. The difference is due to the latency of 5 clocks from inputs $x(n)$ or $y(n-2)$ to the output $y(n)$. Due to the higher latency of 3-stage composite word modules, $x(n)$ is supplied one clock in advance, and three digit registers are used instead of four for synchronization.

The number of gates required for the array is reduced slightly if redundant coefficient latches are omitted. Table 3.2 shows the gate counts of arrays without redundant coefficient latches.

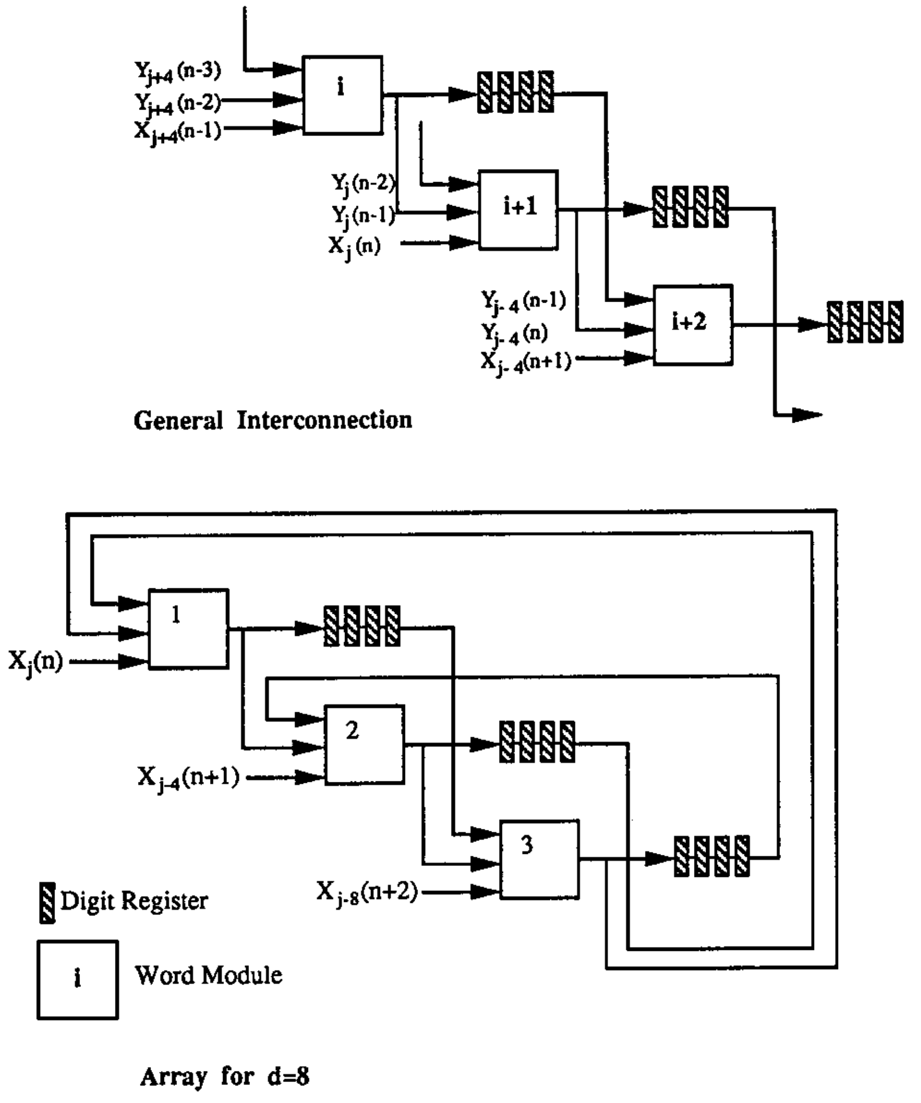


Figure 5.1: Array of 2-Stage Composite Word Modules ($d = 8$)

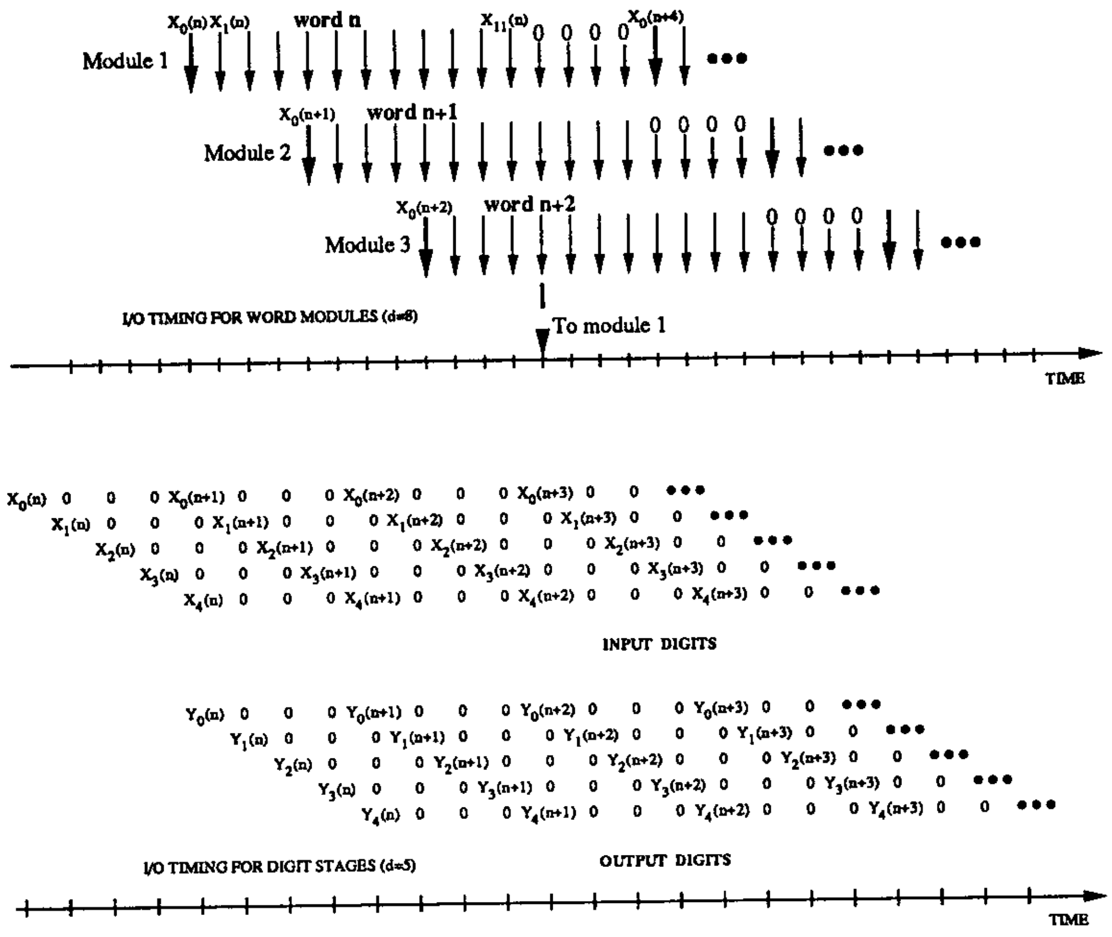


Figure 5.2: I/O Timing for Array of 2-Stage Composite Word Modules ($d = 8$) and for Array of Composite Digit Stages ($d = 5$)

5.2 Array of MA Word Modules

An array of MA word modules is derived from an array of composite word modules by substituting each composite word module with three MA word modules. MA modules were developed in Chapter 4, for $\rho = 2$ and $\rho = 3$. For both modules $\delta_{imp} = 4$. This section discusses the structure, synchronization, performance, and cost of arrays of MA word modules.

Figure 5.3 shows an array of MA word modules for $d = 8$. The structure of the array is a ring, though drawn as a mesh for convenience. The ring has three sets of MA modules, each set having three MA modules. The number of modules required for maximum rate is given by $3\lceil\frac{\delta+d}{\delta_{imp}}\rceil$. The maximum utilization is obtained when $d + \delta$ is an integer multiple of δ_{imp}. The array has three digit inputs and three digit outputs, similar to the array of composite word modules (Figure 5.1).

The synchronization of the MA modules in the array is illustrated in Figure 5.3 by the arrows and the time scale. The digit inputs and outputs are marked by arrows. The relative time is found by reading the time scale corresponding to the arrow tips. Input and output digits are skewed by four clocks (δ_{imp}), similar to the array of composite modules. The latency of the MA word module array is higher than that of the composite word module array, 12 clocks compared to 4.

Table 4.3 compares the performance and costs of arrays of MA word Modules and composite word modules for $d = 8$. The MA module array with $\rho = 2$ has the highest rate, with fewer gates than the array composed of MA modules with $\rho = 3$. The array of MA word modules is more regular than the array of composite word modules. It has a larger number of smaller modules and has no synchronizing digit registers.

5.3 Array of Composite Digit Stages

An array of digit stages is obtained by executing in separate stages, the iterations performed in a single stage of a word module. Each stage is dedicated to specific input and output digits. The number of stages is the same as the number of iterations. The structure, synchronization, performance, and cost of arrays of composite digit stages are described next.

Structure

Figure 5.4 shows the block diagram of a single digit stage. This digit stage performs the same computation as the composite word module shown in Figure 3.3. The only difference is that the residual is not fed back, but passed down to the next digit stage. The input and output of a digit stage are dedicated to specific digits of the input word and output word.

Several digit stages connected in a chain form an array of digit stages. Figure 5.4 shows an array for $d = 5$, $\delta = 2$ and $\delta_{imp} = 4$. The digit stages are numbered according to the iteration number or the input digit number, both denoted by j, where $j = 0, 1, 2, \ldots, \delta + d - 1$. To produce d significant digits of output requires $\delta + d$ iterations.

The precision, or width, of the digit stages need not be identical. Significant savings result when each stage has only the minimum number of fraction digits required to produce d digits of significant outputs. Thus, if digit stage 0 requires f fraction bits, then digit stage 1 requires $f-2$ fraction bits, digit stage 2 requires $f-4$ fraction bits, and the last digit stage requires 0 fraction bits. Figure 5.4 indicates the number of fraction bits for each digit stage for an array for $d = 5$.

Synchronization

The synchronization of digits at inputs, outputs, and within the array, is shown by the arrows and the time scale in Figure 5.4. The timing of each digit is found by reading the time scale at the position of the arrow tip.

The first significant output digit, $Y_0(n)$, appears four clocks after $X_0(n)$, since $\delta_{imp} = 4$. As soon as the first digit appears, the computation of $Y_0(n+1)$ begins. The four cascaded digit registers provide the delay between $Y_j(n-1)$ and $Y_j(n-2)$. The input and output digits of the array are in skew-parallel form as illustrated at the bottom of Figure 5.2.

A digit-stage array based on a 3-stage composite module is slightly different from the array of digit stages based on 2-stage composite modules. The only structural modification required is the removal of one digit register from each cascade of four.

Performance and Cost

As Figure 5.2 shows, each digit stage is utilized only once every four clocks. The clock period is identical to the composite module from which the digit stage array is derived. The rate is the same as the corresponding array of word modules.

The costs of the digit stage arrays are estimated from the implementation of the composite word modules. The results are shown in Table 5.1, for $d = 8$ and coefficients with magnitude < 2.

Table 5.1: Performance/Cost for Array of Composite Digit Stages ($d = 8$)

Array Type	ρ	δ_{imp}	t_{clk}	*Max. Rate*	*Gates*
			ns	*(MSmp./s)*	*/Array*
Composite 2-stage	2	5	14	14.3	12640
Composite 2-stage	3	4	14	17.8	12662
Composite 3-stage	2	5	11	18.8	15038
Composite 3-stage	3	4	11	22.7	14642

5.4 Array of MA Digit Stages

The previous section described how an array of composite digit stages is derived from a composite word module. With an MA word module, a similar derivation results in an array of MA digit stages. Such an array performs a multiply-add. To implement Equation (1.1), three MA digit stage arrays have to be connected to form a larger array. This section discusses the structure, synchronization, performance, and cost of an array of MA Digit Stages.

The MA word module from which the array is derived is shown in Figure 4.2. With $\delta_{imp} = 4$ for the MA modules, and $d = 5$, the array of MA digit stages has a topology similar to the array of composite digit stages shown in Figure 5.4. The arrays have similar timing for the input and output digits, as shown at the bottom of Figure 5.2. The differences between the arrays are listed below.

1. Each MA digit stage has only 2 digit inputs, X and U.

2. The output of each MA digit stage produces $cx + u$.

3. No synchronizing digit registers are required for the array of MA digit stages.

4. The clock rate of the MA digit stages is higher because the MA recurrence is faster (compare clock rates in Table 4.3).

Figure 5.5 shows an array which is a cascade of three MA digit stage arrays. The arrays are connected with a skew of δ. The utilization of each MA digit stage is 0.25, evident from the timing diagram at the bottom of Figure 5.2. The latency is 12 clocks, identical to the array of MA word modules developed in Section 5.3.

A regular array is obtained if all the digit stages are identical. Having identical arrays requires a larger number of gates, but does not change the topology or timing of the array. Regularity may be sacrificed to achieve a design with fewer gates. The number of gates is reduced as follows.

1. The coefficient latches are omitted from all but the three DS0 digit stages.
2. Each digit stage has a different width, diminishing by 2 down the diagonal (Figure 5.5).
3. The first two digit stages do not require a full CPA because the output digit is zero.
4. The last two digit stages do not require the multiplication selectors because the input digits are zero. Like the first two stages, full CPAs are not required since some fraction bits are zero.

The performance and cost of the array of MA digit stages for $\rho = 2$ is given in Table 5.2. The next section compares the arrays.

5.5 Performance and Cost

The performance and costs of the arrays described in preceeding sections of this chapter are summarized in Table 5.2. The gate counts for all arrays assume only one set of coefficient latches. In addition, the gate counts for the digit stage arrays include the reductions listed in the previous section.

The rate and number of gates for the arrays are based on the implementations of the composite word modules in LCA10000 Array technology [LSI87]. The maximum clock rates of the digit stage arrays are the same as the clock rate for the corresponding word module, because the critical paths for the digit stages are the same as the critical paths in the single stage of the word module.

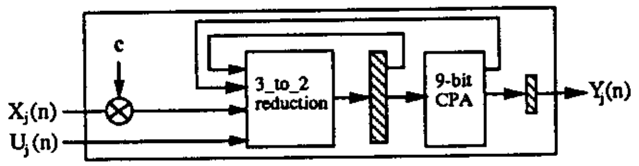
To estimate the number of gates for the arrays of digit stages requires the gate counts for:

- Word modules, for a specific number of fraction slices
- Each fraction slice
- CPA
- Coefficient latches
- Digit register.

Table 5.2 compares arrays for $d = 8$, with coefficient magnitudes < 2, and based on modules with $\delta_{imp} = 4$. The rates for the word module arrays and the corresponding digit stage arrays are identical. The arrays of word modules are smaller than the digit stage arrays. However, utilization of the arrays of digit stages is only 0.25. The next chapter shows how scattered lookahead and recursive doubling techniques improve the utilization and rate of the digit stage arrays.

Table 5.2: Comparison of Performance/Cost of Arrays ($d = 8$)

Array Type	ρ	δ_{imp}	t_{clk} *(ns)*	*Max. Rate (MSmp./s)*	*Gates /Array*
Composite 2-stage WM	3	4	14	17.8	6798
Composite 3-stage WM	3	4	11	22.7	7932
MA WM	2	4	9	27.8	10167
Composite 2-stage DS	3	4	14	17.8	12662
Composite 3-stage DS	3	4	11	22.7	14642
MA DS	2	4	9	27.8	21054



Schematic of MA Module (y=u+cx)

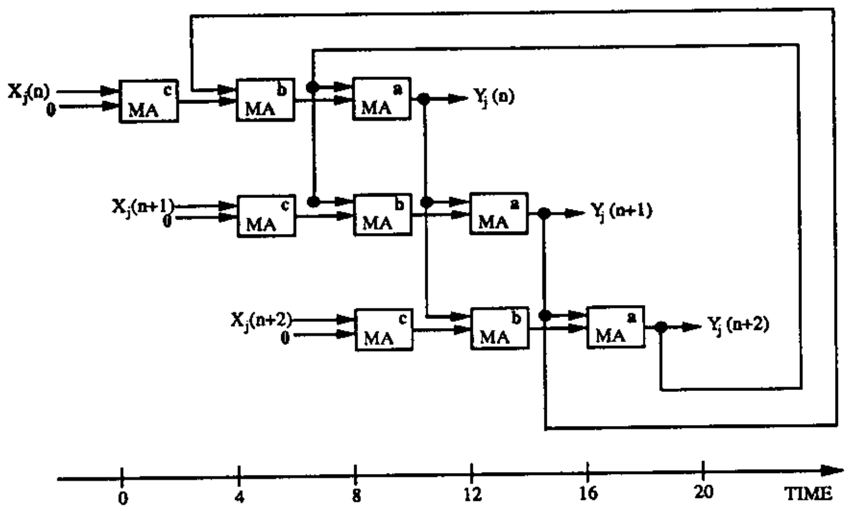


Figure 5.3: Array of MA Word Modules ($d = 8$)

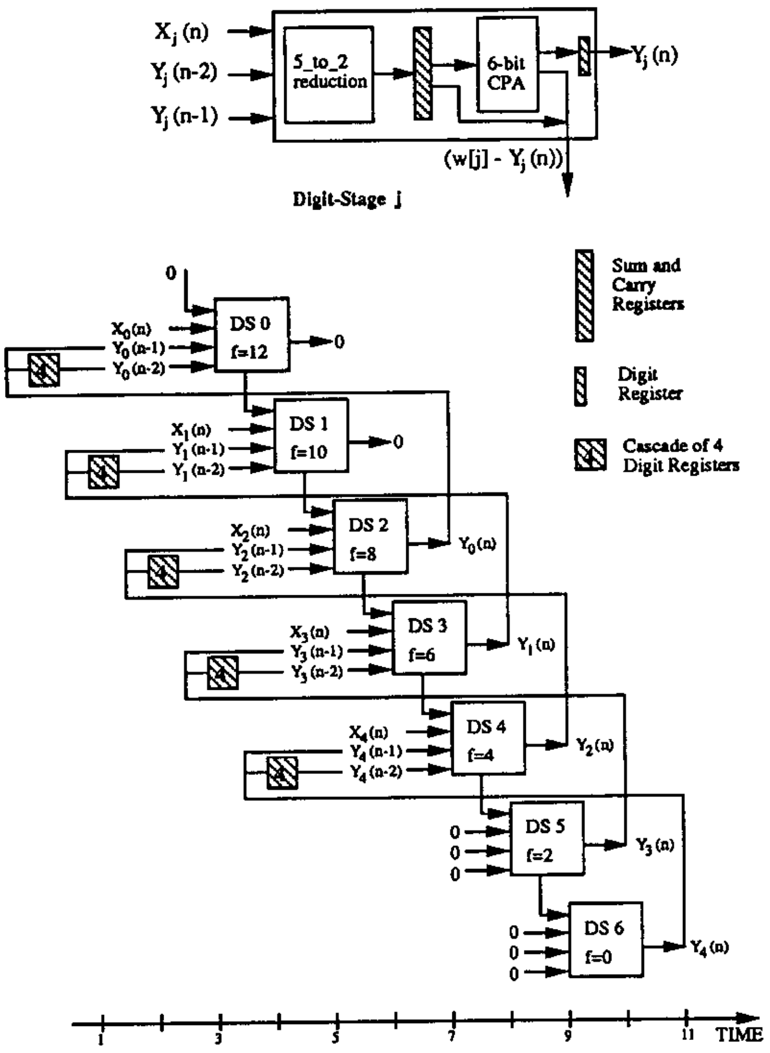


Figure 5.4: Array of Composite Digit Stages ($d = 5$)

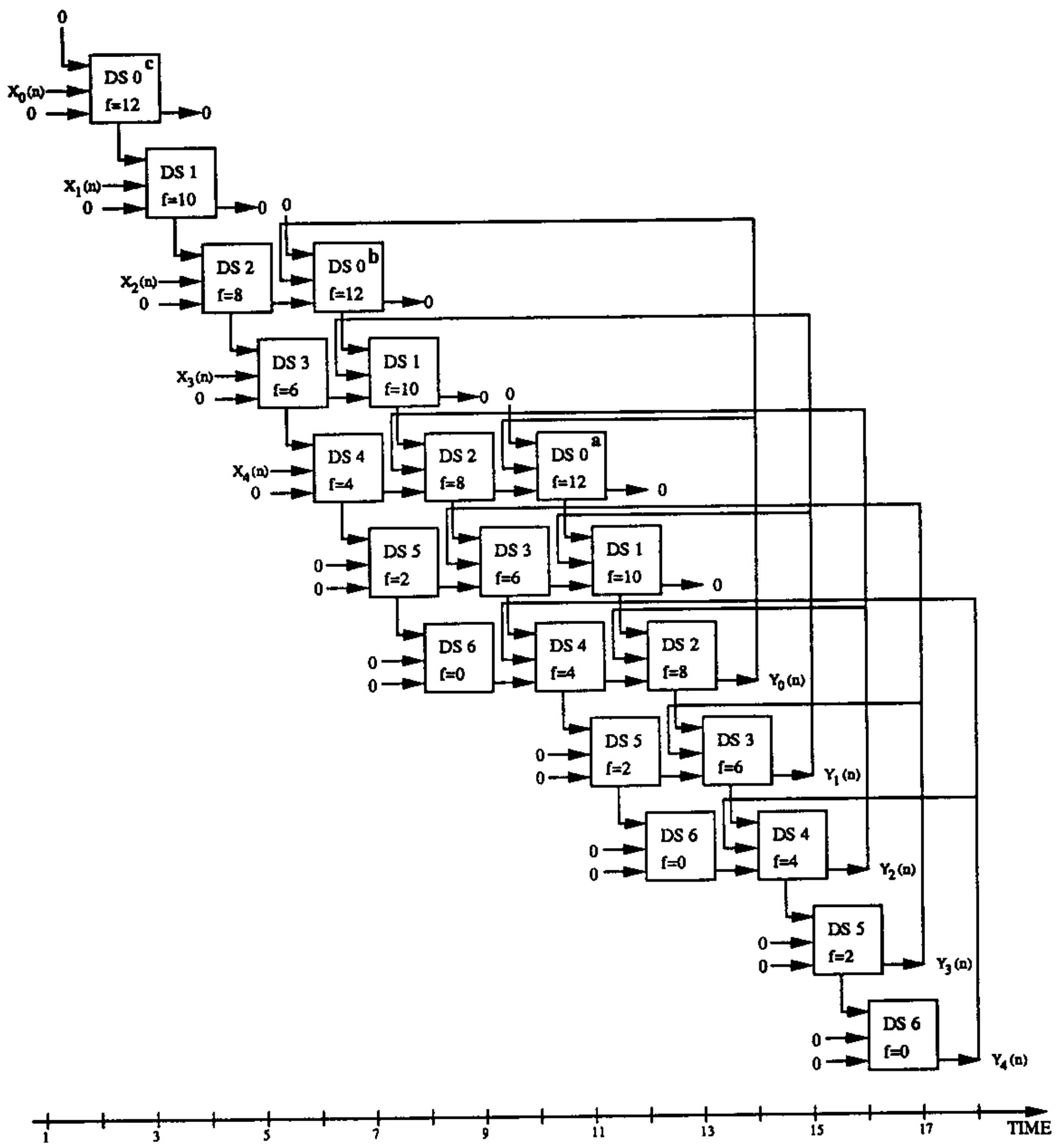


Figure 5.5: Array of MA Digit Stages ($d = 5$)

CHAPTER 6

Algorithmic Transformations for Higher Performance

The previous chapters described on-line schemes that implement expression (1.1). This chapter describes two transformations, the Recursive Unfolding (RU) method [Kog81] in Section 6.1 and the Scattered Lookahead (SL) method [PrMs87a] in Section 6.2. The performance and cost of on-line arrays using the superior SL method are described in Section 6.3. The arrays are composed of modules developed in previous chapters. Section 6.3 also compares the on-line implementations with a conventional implementation using SL (Appendix A). The descriptions of the methods and the implementations focus primarily on the IIR component of the computation.

Both RU and SL transform the difference Equation (1.1) to produce an equivalent expression that allows higher performance. The higher performance is achieved by removing the dependency of $y(n)$ on the immediately preceeding output $y(n-1)$. An important factor to be considered in a transformation is the stability of the resulting expression. Since stability discussions consider the poles of the corresponding transfer functions, the difference Equation (1.1) is shown as a transfer function in (6.1).

$$H(z) = \frac{c}{1 - az^{-1} - bz^{-2}} \tag{6.1}$$

6.1 Recursive Unfolding Method

Recursive unfolding (RU) is a method somewhat similar to recursive doubling [Kog73]. It is also referred to as back-up [Kog81] or lookahead. This section describes the RU method and the stability of the resulting expressions. Performance and cost of the RU method is compared to the SL method in Section 6.3.

To perform RU on Equation (1.1), $y(n-1)$ is substituted in terms of $y(n-2)$ and $y(n-3)$ using expression (1.1):

$$y(n-1) = a \cdot y(n-2) + b \cdot y(n-3) + c \cdot x(n-1) \tag{6.2}$$

As a result of the substitution, $y(n)$ is independent of $y(n-1)$ and depends on $y(n-2)$ and $y(n-3)$ as (6.3) shows. The $x(n-i)$ terms in (6.3) are denoted by the FIR expression $f_1(n)$.

$$y(n) \quad = \quad (a^2+b)\cdot y(n-2) + ab\cdot y(n-3) + f_1(n) \tag{6.3}$$

Expression (6.3) is obtained after applying *1 level* of RU. Using (6.3) instead of (1.1) enables simultaneous computation of two values, $y(n)$ and $y(n-1)$, doubling the potential maximum rate.

Second and Third Levels of RU

The RU process can be performed again on (6.3) to remove the dependency on the $y(n-2)$ term. After the second level of RU, $y(n)$ depends on $y(n-3)$ and previous values of y. To obtain an expression that potentially quadruples the rate of (1.1), the second and third level of RU are applied to (1.1) as follows.

1. Substitute $y(n-2)$ in (6.3), expressing $y(n-2)$ using (6.3).
2. Substitute $y(n-3)$ in (6.3), expressing $y(n-3)$ using (6.2).

The resulting expression is (6.4), and $f_3(n)$ denotes an FIR expression including terms $x(n)$, $x(n-1)$, ..., $x(n-3)$. If the numerator of (6.1) is second-order, the FIR term will include $x(n-4)$ and $x(n-5)$ terms as well. If step 2 above expressed $y(n-3)$ using (6.3), then the terms $y(n-4)$, $y(n-5)$ and $y(n-6)$ appear in the resulting expression. Computing an expression with three y terms on the right hand side is slower and costlier than computing one with two terms.

$$\begin{aligned} y(n) \quad &= \quad (a^4+3a^2b+b^2)\cdot y(n-4) \\ &\quad +(a^3b+2ab^2)\cdot y(n-5) + f_3(n) \end{aligned} \tag{6.4}$$

Stability of RU

To discuss stability of expressions obtained by applying RU, the poles of the corresponding transfer functions must be determined. The first level of RU introduces a cancelling pole-zero pair to (6.1) as shown in (6.5).

$$H(z) \quad = \quad \frac{c(1+az^{-1})}{(1-az^{-1}-bz^{-2})(1+az^{-1})} \tag{6.5}$$

The second and third levels of RU introduce pole-zero pairs expressed by:

$$H(z) = \frac{c(1 + az^{-1} + (a^2 + b)z^{-2} + (a^3 + 2ab)z^{-3})}{(1 - az^{-1} - bz^{-2})(1 + az^{-1} + (a^2 + b)z^{-2} + (a^3 + 2ab)z^{-3})} \tag{6.6}$$

The stability of the original transfer function (6.1) is guaranteed when the coefficients are within the stability triangle as shown in Figure 7.1. Thus, $|a| < 2$ and $|b| < 1$. The pole introduced by the first level of RU is inside the unit circle in the z-plane when $|a| < 1$. Thus, for $a > 1$ the pole is unstable. The poles introduced by the second and third levels of RU can be found if a and b are known. The next section describes the scattered lookahead method which has superior stability characteristics.

6.2 Scattered Lookahead Method

The scattered lookahead (SL) method, due to Parhi and Messerschmitt, was originally intended to allow pipelining of conventional implementations of IIR filters [PrMs87a]. The SL transformation is useful for on-line implementations as well. Potential throughput is doubled with each level of scattered lookahead. The advantage over RU is the stability of the poles introduced in the process. With SL, the output depends on previous values *scattered* in time. The following description is based on [PrHt88].

First Level of SL

The SL method is based on adding cancelling pole-zero pairs to the original transfer function (6.1). The poles of the original transfer function are assumed to be complex conjugates, given by $Re^{j\theta}$ and $Re^{-j\theta}$. Substituting $2R\cos\theta = a$ and $R^2 = -b$, the original transfer function is written as:

$$H(z) = \frac{c}{1 - 2R\cos\theta z^{-1} + R^2 z^{-2}} \tag{6.7}$$

For the first level of SL, two canceling pole-zero pairs, $(1 + 2R\cos\theta z^{-1} + R^2 z^{-2})$, are added. The transformed expression is given by (6.8), and the corresponding difference equation by (6.9). The FIR terms in (6.9) are denoted by $v_1(n)$.

$$H(z) = \frac{c(1 + 2R\cos\theta z^{-1} + R^2 z^{-2})}{1 - 2R^2\cos 2\theta z^{-2} + R^4 z^{-4}} \tag{6.8}$$

$$y(n) = 2R^2\cos 2\theta \cdot y(n-2) - R^4 \cdot y(n-4) + v_1(n) \tag{6.9}$$

Second Level of SL

The second level of SL is applied by adding the pole-zero pairs $(1+2R^2\cos 2\theta z^{-2}+R^4z^{-4})$ to (6.8). The resulting transfer function is given by (6.10), and the corresponding difference equation is shown in (6.11).

$$H(z) = \frac{c(1+2R\cos\theta z^{-1}+R^2z^{-2})(1+2R^2\cos 2\theta z^{-2}+R^4z^{-4})}{1-2R^4\cos 4\theta z^{-4}+R^8z^{-8}} \tag{6.10}$$

$$y(n) = 2R^4\cos 4\theta\cdot y(n-4) - R^8\cdot y(n-8) + v_2(n) \tag{6.11}$$

The FIR expression $v_2(n)$ consists of terms $x(n)$, $x(n-1)$, ..., $x(n-6)$. If (6.1) had a second-order numerator (two zeros outside the origin), $x(n-7)$ and $x(n-8)$ terms will be included in the FIR expression. The expressions for $v_1(n)$ and $v_2(n)$ for the case $c=1$ can be derived from (B.2) and (B.3) in Appendix B. The next section compares the stability of the SL method with that of the RU method.

Stability Comparison

The main difference between the RU method and the SL method is in the magnitude of the added poles. In the SL method, both levels add poles with magnitude R. Thus, the added poles are no more sensitive (not closer to unit circle) than the original poles.

With respect to the difference equations, stability is reflected in the range of values of the coefficients. With SL, coefficients of the recent y term (i.e., the closest term to the one computed) are always in the range [-2,2], and coefficients of the distant y term are in the region [-1,1]. With 1 level of RU the added pole has magnitude $|a|$. Thus the pole is stable for $|a| < 1$. Table 6.2 shows the ranges of the coefficients for both methods. The next section compares the two methods and gives performance and cost estimates for arrays of on-line modules using lookahead.

6.3 Multi-Module Arrays with Lookahead

The application of lookahead produces arrays that deliver two to four times the performance of arrays without lookahead. This section discusses the performance and cost of arrays that deliver maximum performance. The arrays described here are based on modules and arrays developed in Chapters 3,4 and 5. The arrays are restricted to the computation of the IIR portion of the transfer function and assume that d=8. A reference point for the cost and the performance of on-line arrays is provided by including performance and cost figures for a conventional implementation using scattered lookahead (Appendix A).

Maximum Throughput for General Lookahead

As the previous sections of this chapter showed, applying different levels of RU and SL produce different expressions. To find the maximum througput for such expressions, computed by modules with different δ_{imp}, a general formula is written as follows.

- General recursive expression is

$$y(n) \quad = \quad a_i \cdot y(n-i) + a_j \cdot y(n-j) + u(n) \tag{6.12}$$

- Latencies of the module are: δ_i clocks from $y(n-i)$ to $y(n)$, and δ_j clocks from $y(n-j)$ to $y(n)$.
- Minimum clock period for the module is t_{clk} ns.
- Maximum rate of the module is μ Msamples/sec.

The maximum rate is:

$$\mu \quad = \quad \frac{1000}{t_{clk}} \min\left(\frac{i}{\delta_i}, \frac{j}{\delta_j}\right) \tag{6.13}$$

The values of i and j depend on the expression. Table 6.1 characterizes some modules described previously, in terms of t_{clk}, δ_i, and δ_j.

Table 6.1: Latency of Modules for Lookahead Computation

Module Type	ρ	$t_{clk}(ns)$	δ_i	δ_j
Composite 3-stage	3	11	4	5
MA	2	9	4	8
MA	3	10	4	8

Reduction of Potential Maximum Rate of RU

Section 6.1 showed that 1 level of RU doubles the *potential* rate and that 3 levels of RU quadruples it. The actual rate obtained from RU is less, due to the increased coefficient range and the low value of j, as defined in (6.13). With 1 level of RU the coefficient ranges are [-1,3] and [-2,2]. Table 6.2 indicates the coefficient ranges for SL and RU.

The increased coefficient range does not allow the use of some modules. For example, the MA module with $\rho = 2$ cannot accomodate coefficients with a range of [-1,3]. As Table 6.2 shows, higher levels of RU require even larger coefficients, reducing the gap between actual and potential rates further.

Another cause of rate reduction with RU is the relatively low value of j, as Table 6.2 shows. For 1 level of RU, as (6.13) indicates, the relative rate increase with MA modules is only a factor of 1.5.

Table 6.2: Coefficient Ranges and Max. Rate for SL and RU

Expression	Ranges of Coefficients	i, j	Relative Potential Max. Rate
Original	[-2,2], [-1,1]	1, 2	1
1 Level SL	[-2,2], [-1,1]	2, 4	2
2 Level SL	[-2,2], [-1,1]	4, 8	4
1 Level RU	[-1,3], [-2,2]	2, 3	2
3 Level RU	[0,5], [-4,4]	4, 5	4

Comparison of Performance and Cost

Table 6.3 shows the performance and costs for different arrays with 1 or 2 levels of lookahead. The maximum rates are based on the maximum clock rates of the individual modules. Cost is estimated by the number of gates required for these modules. All numbers for performance and costs are based on implementation of modules in a LCA10000 Array [LSI87].

Table 6.3 indicates the following.

1. Scattered Lookahead produces the highest rate for any level.
2. The most cost-effective *on-line* arrays are based on the 3-stage Composite Module.
3. The most cost effective, *overall*, is the conventional implementation.
4. The on-line MA digit stage array ($\rho = 2$) produces the highest rate.
5. The total cost is dominated by the FIR portion of the computation, particularly with 2 levels of lookahead. The number of FIR terms shown is for a general second-order transfer function with 2 zeros outside the origin.

Note that the computation performed by the arrays is given by (6.12). The 3-stage Composite module was developed to compute expression (1.1). Consequently, the number of gates shown in Table 6.3 is an estimate based on the actual design for computing (1.2), which is an instance of (6.12).

The digit stage arrays are more cost effective with 2 levels of lookahead. The array of on-line MA digit stages for d=8 is similar to the array shown in Figure 5.5 for d=5. The utilization of the array is only 0.25, without lookahead, as Figure 5.2 shows. With each level of lookahead the array utilization doubles, and full utilization is achieved with 2 levels of SL.

The implementations shown in the table vary in degree of modularity. The most modular design is the MA word module array for 1 level of SL. The most irregular design is the conventional implementation (Appendix A). Since silicon area is a more appropriate cost measure than the number of gates, modularity must also be considered in estimating the cost.

Table 6.3: Comparison of Performance/Cost of IIR section with Lookahead ($d = 8$)

Type of Array	ρ	Type of Lookahead	t_{clk} (ns)	Max. Rate Msamples/s	Gates	FIR Terms
MA WM	2	SL-1	9	55.5	13396	5
MA DS	2	SL-1	9	55.5	14036	5
Comp. 3-Stage WM	3	SL-1	11	45.4	12732	5
Comp. 3-Stage DS	3	SL-1	11	45.4	12192	5
MA WM	3	RU-1	10	37.5	14552	4
MA DS	3	RU-1	10	37.5	15136	4
Comp. 3-Stage WM	3	RU-1	11	45.4	12732	4
Comp. 3-Stage DS	3	RU-1	11	45.4	12192	4
MA DS	2	SL-2	9	111	14036	9
Comp. 3-Stage DS	3	SL-2	11	91	12192	9
Conventional	-	SL-2	11	91	11796	9

CHAPTER 7

Limit Cycle Oscillations

This chapter analyzes limit cycle oscillations resulting from roundoff errors in conventional and on-line fixed-point recursive filters. Roundoff errors in filters are introduced when double-precision results from multiplication are reduced to single-precision. Limit cycle oscillations are caused by roundoff errors and appear as small-amplitude periodic oscillations in the output. Several bounds have been derived for the amplitude of limit cycles in second-order sections [DSP2]. The analysis in this chapter proves that when scattered lookahead (SL) is applied, the limit cycle amplitude bound decreases, a result contrary to intuition.

This chapter is arranged as follows. Section 7.1 quantifies roundoff errors in on-line composite and on-line MA modules. Section 7.2 derives a bound for the maximum amplitude of zero-input limit cycles in a second-order section with SL. In Section 7.3 the bound for the SL case is shown to be less than the bound without SL. Section 7.4 compares, for two on-line filters, the derived bounds with simulation results.

7.1 Roundoff Error in On-Line Modules

Conventional multiplication of single-precision values produces a double-precision result. Due to the finite precision of the hardware, results must be reduced to single-precision by rounding. An advantage of on-line multiplication is that it produces only the required digits, beginning at the most significant. Thus, a separate rounding step is not required. The roundoff error, in conventional or on-line multiplication, is the difference between the full-precision result and the single-precision output. For on-line multiplication the roundoff error is the residual after the least significant output digit is produced. Roundoff error is introduced at each point where precision is reduced. On-line or conventional MA modules introduce roundoff error after each multiplication. On-line composite modules introduce roundoff error once every two or three multiplications.

In radix-r on-line modules the roundoff error is $r^{-d}(w[d+\delta-1] - Y_{d+\delta-1})$, the value of the residual after d digits have been produced. The *scaled* value of the residual is in the range $[\underline{\xi}, \overline{\xi}]$, where $\underline{\xi}$ and $\overline{\xi}$ are defined by (3.14) and (3.15). The roundoff error in the composite module and MA module is bounded by

$$-\frac{1}{2}r^{-d} \leq e_{MA} \leq \left(\frac{1}{2}+2^{-6}\right)r^{-d} \tag{7.1}$$

$$-\frac{1}{2}r^{-d} \leq e_{CM} \leq \left(\frac{1}{2}+2^{-3}\right)r^{-d} \tag{7.2}$$

where $r = 4$ and d is the number of fraction digits in the output. For a cascade of MA modules, the roundoff error is the sum of errors produced by each MA module. The additive rule holds because the errors are independent random values. As (7.1) and (7.2) show, the positive roundoff error bounds differ slightly from the perfect rounding case. The roundoff error for the cascaded pair computing a second-order recursion is given by (7.3).

$$-r^{-d} \leq e_{MA2} \leq (1+2^{-5})r^{-d} \tag{7.3}$$

7.2 Bound on Limit Cycle Amplitude

Limit cycles are small-amplitude oscillations that corrupt the output of recursive digital filters. They are caused by roundoff error and may occur with zero or non-zero input. Limit cycles are a significant problem in fixed-point direct-form recursive filters. The *forced limit cycle oscillation* occurs with simple non-zero inputs such as constant or alternating sequences. For complex inputs the forced limit cycle oscillation appears as noise and is not much of a problem [Sam88]. The more serious non-linear effect is the *zero-input limit cycle oscillation*, the focus of discussion in this chapter. Various types of non-linear effects are discussed in [Sam88].

Zero-input limit cycles have been studied extensively for second-order recursive filters [DSP2]. Bounds have been derived for the maximum amplitude of the limit cycle. Two such bounds are the absolute bound by Long and Trick [LnTr73] and the RMS bound by Sandberg and Kaiser [SnKs72]. The basic approach of increasing working precision of *on-line recursive filters* was used by Brackert to *eliminate* all zero-input and forced nonlinear oscillations *without affecting sampling rate* [Bra89b]. Knowing the maximum possible amplitude of the limit cycle allows the filter precision to be increased by the fewest number of bits necessary to contain the entire oscillation.

Absolute Bound for Limit Cycles in a Second-Order Recursion

The following is a summary of the derivation, by Long and Trick [LnTr73], of a bound for the maximum amplitude of periodic limit cycles in a second-order direct-form recursion. In the first part of their derivation, Long and Trick derive an expression for the maximum amplitude of a linear N^{th}-order recursion in terms of the impulse response and limit cycle period. In the second part, the general result is used to derive the bound for a second-order recursion. The summary follows.

Consider the N^{th}-order non-linear difference equation (7.4), where $[\cdot]_r$ denotes the rounding function, and the input or forcing term is zero for all n.

$$y(n) = \sum_{j=1}^{N}[a_j y(n-j)]_r \tag{7.4}$$

The derivation assumes the following:

1. Rounding is done after each multiplication.
2. Equation without rounding is asymptotically stable.
3. Coefficients and $y(n)$ values are fixed-point numbers.
4. Limit cycles are periodic.

Let $e_j(n)$ be the error due to rounding $[a_j y(n-j)]_r$. Thus (7.4) can be converted to a linear equation:

$$y(n) = \sum_{j=1}^{N}(a_j y(n-j) + e_j(n)) \tag{7.5}$$

Defining a single error term $e(n)$, (7.5) can be expressed as (7.7).

$$e(n) = \sum_{j=1}^{N} e_j(n) \tag{7.6}$$

$$y(n) = \sum_{j=1}^{N} a_j y(n-j) + e(n) \tag{7.7}$$

To simplify the derivation, without loss of generality, the quantization error is assumed to be 1. Thus, for perfect rounding $|e_j(n)| \leq \frac{1}{2}$. Since the errors are uncorrelated random values, $e(n)$ is given by:

$$|e(n)| = \frac{N}{2} \tag{7.8}$$

For the on-line MA module $|e(n)| \leq N\bar{\xi}$ and for the on-line composite module $|e(n)| \leq \frac{N\bar{\xi}}{2}$.

For a periodic limit cycle with period M,

$$e(k) \;=\; e(k+M) \qquad (7.9)$$

for all k. Considering the error term $e(n)$ to be an input to the linear system with an impulse response $h(n)$, the output is given by the convolution sum:

$$y(n) \;=\; \sum_{k=-\infty}^{n} h(n-k)e(k) \qquad (7.10)$$

Expressing (7.10) as a double summation gives

$$y(n) \;=\; \sum_{i=0}^{\infty}\left[\sum_{k=n-(i+1)M+1}^{n-iM} h(n-k)e(k)\right] \qquad (7.11)$$

where the terms are in groups of M each. Letting $p = n - iM - k$, equation (7.11) can be written:

$$y(n) \;=\; \sum_{i=0}^{\infty}\left[\sum_{p=0}^{M-1} h(p+iM)e(n-iM-p)\right] \qquad (7.12)$$

Substituting $e(n-p)$ for $e(n-iM-p)$ and interchanging the summations, (7.12) can be written:

$$y(n) \;=\; \sum_{p=0}^{M-1} e(n-p)\left[\sum_{i=0}^{\infty} h(p+iM)\right] \qquad (7.13)$$

The bound for the absolute value of the amplitude of the limit cycle of period M is given by (7.14) for all n.

$$|y(n)|_{max} \;=\; \frac{N}{2}\sum_{p=0}^{M-1}\left|\sum_{i=0}^{\infty} h(p+iM)\right| \qquad (7.14)$$

Thus, given an explicit form of $h(p)$, (7.14) can be used to derive an expression for $|y(n)|_{max}$. To obtain an expression for $h(p)$, the inverse z-transform, given by (7.15), must be evaluated. The evaluation can be done using the residue theorem or by using the simpler partial-fraction expansion method [PrMn88].

$$h(p) \;=\; \frac{1}{2\pi j}\oint H(z)z^{(p-1)}dz \qquad (7.15)$$

The partial-fraction expansion method is used here to derive the same bound obtained in [LnTr73]. For a second-order recursion defined by (7.16), the corresponding partial product form for *distinct roots* (i.e., $q \neq 0$) is given by (7.17).

$$H(z) = \frac{z^2}{z^2 - az - b} \tag{7.16}$$

$$\frac{H(z)}{z} = \frac{1}{2q}\left(\frac{a_2 + q}{z - (a_2 + q)} - \frac{a_2 - q}{z - (a_2 - q)}\right) \tag{7.17}$$

$$a_2 \overset{def}{=} \frac{a}{2} \tag{7.18}$$

$$q \overset{def}{=} \sqrt{\left(\frac{a}{2}\right)^2 + b} \tag{7.19}$$

Once $h(p)$ is obtained by using the transform relation $\frac{1}{1-pz^{-1}} \Leftrightarrow p^n$, then $h(p + iM)$ follows.

$$h(p) = \frac{1}{2q}(a_2 + q)^{p+1} - \frac{1}{2q}(a_2 - q)^{p+1} \tag{7.20}$$

$$h(p + iM) = \frac{1}{2q}(a_2 + q)^{p+1+iM} - \frac{1}{2q}(a_2 - q)^{p+1+iM} \tag{7.21}$$

Finally, $\sum_{i=0}^{\infty} h(p+iM)$ is obtained, and, using (7.14) with $N = 2$, the desired bound is also obtained.

$$\sum_{i=0}^{\infty} h(p + iM) = \frac{1}{2q}\left[\frac{(a_2 + q)^{p+1}}{1 - (a_2 + q)^M} - \frac{(a_2 - q)^{p+1}}{1 - (a_2 - q)^M}\right] \tag{7.22}$$

$$|y(n)|_{max} = \frac{1}{|2q|}\sum_{p=0}^{M-1}\left|\frac{(a_2 + q)^{p+1}}{1 - (a_2 + q)^M} - \frac{(a_2 - q)^{p+1}}{1 - (a_2 - q)^M}\right| \tag{7.23}$$

The bound derived is for distinct roots, the case for useful second-order filter sections. The bound for coincident roots is given by (7.24). The derivation is simpler than for the distinct root case and is omitted for brevity.

$$|y(n)|_{max} = \sum_{p=0}^{M-1}\left|a_2^p\left[\frac{p}{1 - a_2^M} + \frac{1 + a_2^M(M-1)}{(1 - a_2^M)^2}\right]\right| \tag{7.24}$$

Given the filter coefficients a and b, the bound can be calculated for any desired period M. Since the period is not known in advance, the bound has to be computed for all values of M.

Absolute Bound for Second-Order Recursion with SL

This section extends the Long and Trick method to derive the absolute bound for a second-order recursion transformed by scattered lookahead. This bound can then be compared with the bound without SL to determine the effect SL has on zero-input limit cycles. The derivation begins with the transformed expression (7.25). The two zeros introduced by the SL transformation are omitted from (7.25) because the input is zero. The partial-fraction expansion of $H_{SL}(z)$ for distinct roots ($q \neq 0$) is shown in (7.26).

$$H_{SL}(z) = \frac{z^4}{(z^2 - az - b)(z^2 + az - b)} \tag{7.25}$$

$$\begin{aligned}\frac{H_{SL}(z)}{z} &= \frac{1}{4qa}\left[\frac{(a_2+q)^2}{z-(a_2+q)} - \frac{(a_2-q)^2}{z-(a_2-q)}\right.\\ &\left. - \frac{(-a_2+q)^2}{z-(-a_2+q)} + \frac{(-a_2-q)^2}{z-(-a_2-q)}\right]\end{aligned} \tag{7.26}$$

Note that a_2 and q are defined by (7.18) and (7.19). Applying the inverse z-transform to (7.26) produces $h_{SL}(p)$, from which $h_{SL}(p+iM)$ is obtained.

$$\begin{aligned}h_{SL}(p) &= \frac{1}{4qa}\left[(a_2+q)^{p+2} - (a_2-q)^{p+2}\right.\\ &\left. -(-a_2+q)^{p+2} + (-a_2-q)^{p+2}\right]\end{aligned} \tag{7.27}$$

$$\begin{aligned}h_{SL}(p+iM) &= \frac{1}{4qa}\left[(a_2+q)^{p+2+iM} - (a_2-q)^{p+2+iM}\right.\\ &\left. -(-a_2+q)^{p+2+iM} + (-a_2-q)^{p+2+iM}\right]\end{aligned} \tag{7.28}$$

To obtain the bound, the expression for $h_{SL}(p+iM)$ is substituted in (7.14) and summed over i to give (7.29).

$$\begin{aligned}\sum_{i=0}^{\infty} h_{SL}(p+iM) &= \frac{1}{4qa}\left[\frac{(a_2+q)^{p+2}}{1-(a_2+q)^M} - \frac{(a_2-q)^{p+2}}{1-(a_2-q)^M}\right.\\ &\left. - \frac{(-a_2+q)^{p+2}}{1-(-a_2+q)^M} + \frac{(-a_2-q)^{p+2}}{1-(-a_2-q)^M}\right]\end{aligned} \tag{7.29}$$

The bound $|y_{SL}(n)|_{max}$ is now reduced to a single summation over M, as shown in (7.30). Given M and the filter coefficients, (7.30) can be used to find the bound for the limit cycle for distinct roots.

$$\begin{aligned}|y_{SL}(n)|_{max} &= \frac{1}{|4qa|}\sum_{p=0}^{M-1}\left|\frac{(a_2+q)^{p+2}}{1-(a_2+q)^M} - \frac{(a_2-q)^{p+2}}{1-(a_2-q)^M}\right. \\ &\quad \left. - \frac{(-a_2+q)^{p+2}}{1-(-a_2+q)^M} + \frac{(-a_2-q)^{p+2}}{1-(-a_2-q)^M}\right| \qquad (7.30)\end{aligned}$$

If the filter coefficients satisfy $a^2 = -4b$, then the roots are coincident and (7.30) cannot be used. For coincident roots, the partial-fraction expansion is given by (7.31). The derivation is easier than the distinct root case and proceeds similarly. Only the main steps are outlined below.

$$\frac{H_{SL}(z)}{z} = \frac{\frac{1}{2}}{z-a_2} + \frac{\frac{1}{2}}{z+a_2} + \frac{\frac{a}{8}}{(z-a_2)^2} - \frac{\frac{a}{8}}{(z+a_2)^2} \qquad (7.31)$$

$$\begin{aligned}h_{SL}(p+iM) &= \frac{1}{2}(-a_2)^{p+iM} + \frac{1}{2}a_2^{p+iM} + \left(\frac{p+iM}{4}\right)(-a_2)^{p+iM} \\ &\quad - \left(\frac{p+iM}{4}\right)a_2^{p+iM} \qquad (7.32)\end{aligned}$$

$$\begin{aligned}|y_{SL}(n)|_{max} &= \frac{1}{4}\sum_{p=0}^{M-1}\left|(p+2)\left[\frac{(-a_2)^p}{1-(-a_2)^M} + \frac{(a_2)^p}{1-(a_2)^M}\right]\right. \\ &\quad \left. +M\left[\frac{(-a_2)^p}{[1-(-a_2)^M]^2} - \frac{(a_2)^p}{[1-(a_2)^M]^2}\right]\right| \qquad (7.33)\end{aligned}$$

The amplitude bound of the limit cycle for the coincident root case is given by (7.33).

7.3 Comparison of Limit Cycle Amplitude Bounds

The amplitude bounds given by expressions (7.30) and (7.23), have to be compared to find how SL affects zero-input limit cycle amplitude. The comparison is made for the case where roots of the second-order recursion are distinct, which includes all useful second-order filter sections. Expressions (7.30) and (7.23) give the bounds for the recursion with and without SL. The comparison shows that the amplitude bound of limit cycles with SL is less than the bound for the recursion without SL when $|a| \geq 0.74$. The result is obtained by proving four theorems.

Theorem 1 *For large periods, scattered lookahead reduces the amplitude bound of limit cycles in recursions with distinct roots by a factor γ.*

$$\begin{aligned} |y_{SL}(n)|_{M\to\infty} &< \tfrac{1}{\gamma}|y(n)|_{M\to\infty} \\ \gamma &\overset{def}{=} \tfrac{|a|(1+|a|+|b|)}{1+|b|} \end{aligned}$$

Proof

Consider the bound for the distinct root case with SL given by (7.30). For large M, $(\pm a_2 \pm q)^M \ll 1$ since $|\pm a_2 \pm q| < 1$. Thus, for large M (7.30) reduces to:

$$\begin{aligned} |y_{SL}(n)|_{M\to\infty} = \frac{1}{|4qa|} \sum_{p=0}^{M-1} \Big|(a_2+q)^{p+2} - (a_2-q)^{p+2} \\ -(-a_2+q)^{p+2} + (-a_2-q)^{p+2}\Big| \end{aligned} \tag{7.34}$$

For odd values of p the terms in the summation are zero, leaving only the terms for even values of p. This simplifies the bound to:

$$|y_{SL}(n)|_{M\to\infty} = \frac{1}{|2qa|} \sum_{p=0,even}^{M-1} \left|(a_2+q)^{p+2} - (a_2-q)^{p+2}\right| \tag{7.35}$$

Using the following notation for brevity,

$$u \overset{def}{=} a_2 + q \tag{7.36}$$

$$v \overset{def}{=} a_2 - q \tag{7.37}$$

$$w^i \overset{def}{=} u^i - v^i \tag{7.38}$$

the summation in (7.35) can be expanded to give

$$|y_{SL}(n)|_{M\to\infty} = \frac{1}{|2qa|}\left[|w^2| + |w^4| + |w^6| + \ldots + |w^{M+1}|\right] \tag{7.39}$$

where M is assumed, without loss of generality, to be a large odd value.

Now consider (7.23), the bound for the second-order recursion without SL. For large M the bound reduces to:

$$|y(n)|_{M\to\infty} = \frac{1}{|2q|} \sum_{p=0}^{M-1} \left|(a_2+q)^{p+1} - (a_2-q)^{p+1}\right| \tag{7.40}$$

The summation in (7.40) expressed in terms of w gives the following.

$$|y(n)|_{M\to\infty} = \frac{1}{|2q|}\left[|w| + |w^2| + |w^3| + \ldots + |w^M|\right] \tag{7.41}$$

Comparing the two series (7.39) and (7.41), it is clear that all terms in the former are included in the latter, except the last term when M is odd-valued. As M becomes large the last term vanishes. It is possible to express one in terms of the other by proceeding as follows. A simple factorization shows that

$$w^{p+1} = aw^p + bw^{p-1} \tag{7.42}$$

which gives the following inequality.

$$|w^{p+1}| + |b||w^{p-1}| \geq |a||w^p| \tag{7.43}$$

Evaluating (7.43) with $p = 2, 4, 6, \ldots$, and adding each side of the inequality produces the following,

$$\left[|w^3| + |w^5| + |w^7| + \ldots\right] + |b|\left[|w| + |w^3| + |w^5| + \ldots\right] > |a|\left[|w^2| + |w^4| + |w^6| + \ldots\right] \tag{7.44}$$

and, adding $|w|$ to the left hand side of 7.44 gives 7.45.

$$(1+|b|)\left[|w| + |w^3| + |w^5| + \ldots\right] > |a|\left[|w^2| + |w^4| + |w^6| + \ldots\right] \tag{7.45}$$

Using this result, the series $[|w|+|w^3|+|w^5|+\ldots]$ can be replaced by $\frac{|a|}{1+|b|}[|w^2| + |w^4| + |w^6| + \ldots]$ in (7.41). The latter series itself can be expressed in terms of $|y_{SL}(n)|_{M\to\infty}$. These substitutions give (7.46), a relationship between the bounds for large M.

$$|y(n)|_{M\to\infty} > \left[\frac{|a|(1+|a|+|b|)}{1+|b|}\right]|y_{SL}(n)|_{M\to\infty} \tag{7.46}$$

Substituting the definition of γ given earlier yields the desired relationship.

□

The region in which SL reduces the bound includes all useful filters with $|a| \geq 0.74$ and is shown in Figure 7.1. The value of γ is bounded by $4 \geq \gamma \geq 1$ in the shaded region in Figure 7.1.

Theorem 2 *For the distinct root case with scattered lookahead, the asymptotic bound ($M \rightarrow \infty$) is at least as large as the bound for any finite period of the limit cycle.*

$$|y_{SL}(n)|_{M\rightarrow\infty} \geq |y_{SL}(n)|_M$$

Proof

Consider the bound for distinct roots with SL given by (7.30). For *even-valued M* the bound simplifies to (7.47), where all terms in the summation are zero when p is odd-valued. For *odd-valued M* the bound simplifies to (7.48).

$$|y_{SL}(n)|_{max} = \frac{1}{|2qa|}\sum_{p=0,even}^{M-2}\left|\frac{(a_2+q)^{p+2}}{1-(a_2+q)^M} - \frac{(a_2-q)^{p+2}}{1-(a_2-q)^M}\right| \tag{7.47}$$

$$\begin{aligned}|y_{SL}(n)|_{max} = {} & \frac{1}{|4qa|}\sum_{p=0}^{M-1}\left|\frac{(a_2+q)^{p+2}}{1-(a_2+q)^M} - \frac{(a_2-q)^{p+2}}{1-(a_2-q)^M}\right. \\ & \left. - \frac{(-a_2+q)^{p+2}}{1+(a_2-q)^M} + \frac{(-a_2-q)^{p+2}}{1+(a_2+q)^M}\right|, \quad M\text{=odd}\end{aligned} \tag{7.48}$$

Consider the bound for odd-valued M. The summation in (7.48) can be simplified by considering separately the terms for odd-valued p and even-valued p. This allows the first and last terms in (7.48) to be combined. The second and third terms can also be combined to give the following equivalent expression.

$$|y_{SL}(n)|_{max} = \frac{1}{|4qa|}\sum_{p=0}^{M-1}\begin{cases} P_e(p), & p = \text{even} \\ P_o(p), & p = \text{odd}\end{cases} \tag{7.49}$$

where $P_e(p)$ and $P_o(p)$ are given by the following.

$$\begin{aligned} P_e(p) &= \left|\frac{2(a_2+q)^{p+2}}{1-(a_2+q)^{2M}} - \frac{2(a_2-q)^{p+2}}{1-(a_2-q)^{2M}}\right| \\ P_o(p) &= \left|\frac{2(a_2+q)^{p+M+2}}{1-(a_2+q)^{2M}} - \frac{2(a_2-q)^{p+M+2}}{1-(a_2-q)^{2M}}\right| \end{aligned}$$

The powers in the numerators consist of the even numbers in the range 2 to $2M-2$. Thus the summation limit can be modified to give (7.50), a simpler expression.

$$|y_{SL}(n)|_{max} = \frac{1}{|2qa|}\sum_{p=0,even}^{2M-2}\left|\frac{(a_2+q)^{p+2}}{1-(a_2+q)^{2M}} - \frac{(a_2-q)^{p+2}}{1-(a_2-q)^{2M}}\right| \tag{7.50}$$

Compare (7.47) and (7.50), the bounds for even-valued M and odd-valued M. Since M is an arbitrary value, M in (7.47) can be replaced by $2M$. Thus (7.47) and (7.50) are equivalent, and the proof applies to odd and even M.

Adopting the u and v notation for simplicity, (7.50) can be expanded to give

$$\begin{aligned}|y_{SL}(n)|_{max} = \frac{1}{|2qa|}&\left[\left|\frac{u^2}{1-u^{2M}} - \frac{v^2}{1-v^{2M}}\right| + \left|\frac{u^4}{1-u^{2M}} - \frac{v^4}{1-v^{2M}}\right|\right.\\ &+ \left|\frac{u^6}{1-u^{2M}} - \frac{v^6}{1-v^{2M}}\right| + \ldots + \left|\frac{u^{1+M}}{1-u^{2M}} - \frac{v^{1+M}}{1-v^{2M}}\right|\\ &+ \left|\frac{u^{3+M}}{1-u^{2M}} - \frac{v^{3+M}}{1-v^{2M}}\right| + \left|\frac{u^{5+M}}{1-u^{2M}} - \frac{v^{5+M}}{1-v^{2M}}\right|\\ &\left.+ \left|\frac{u^{7+M}}{1-u^{2M}} - \frac{v^{7+M}}{1-v^{2M}}\right| + \ldots + \left|\frac{u^{2M}}{1-u^{2M}} - \frac{v^{2M}}{1-v^{2M}}\right|\right]\end{aligned} \tag{7.51}$$

To produce a sum of infinite series, the denominator terms are replaced by their equivalent infinite series.

$$\frac{1}{1-u^{2M}} = 1 + u^{2M} + u^{4M} + u^{6M}\ldots \tag{7.52}$$

$$\frac{1}{1-v^{2M}} = 1 + v^{2M} + v^{4M} + v^{6M}\ldots \tag{7.53}$$

For convenience in representation, each $|\cdot|$ term in (7.51) is replaced by an infinite series in terms of w using (7.52) and (7.53).

$$\begin{aligned}|y_{SL}(n)|_{max} = \frac{1}{|2qa|}&\left[\left|w^2 + w^{2M+2} + w^{4M+2} + w^{6M+2} + \ldots\right|\right.\\ &+ \left|w^4 + w^{2M+4} + w^{4M+4} + w^{6M+4} + \ldots\right|\\ &+ \left|w^6 + w^{2M+6} + w^{4M+6} + w^{6M+6} + \ldots\right|\\ &\quad\vdots\\ &\left.+ \left|w^{2M} + w^{2M+2M} + w^{4M+2M} + w^{6M+2M} + \ldots\right|\right]\end{aligned} \tag{7.54}$$

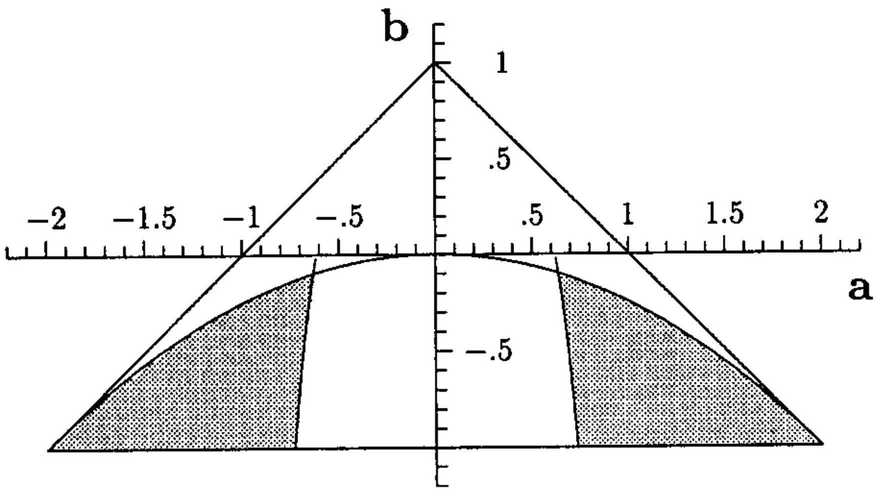


Figure 7.1: Triangle of Stability and Region (shaded) of Complex Roots with Reduced Limit Cycle Amplitude Bounds for SL

The asymptotic bound for the SL case is given by (7.39) which is written in terms of w for comparison.

$$|y_{SL}(n)|_{M\to\infty} = \frac{1}{|2qa|}\left[\left|w^2\right| + \left|w^4\right| + \left|w^6\right| + \ldots\right] \tag{7.55}$$

Comparing (7.54) and (7.55) it is clear that all terms in (7.54) are included in (7.55) and vice versa. The difference is in taking the absolute values. Generalizing the inequality $|a| + |b| \geq |a + b|$, it is obvious that $|y_{SL}(n)|_{M\to\infty} \geq |y_{SL}(n)|_M$.

□

Theorem 3 *For the distinct root case without scattered lookahead, the asymptotic bound (i.e., as $M \to \infty$) is at least as large as the bound for any finite period of the limit cycle.*

$$|y(n)|_{M\to\infty} \geq |y(n)|_M$$

The proof is similar to Theorem 2.

□

Theorem 4 *For all even-valued periods, scattered lookahead reduces the amplitude bound of limit cycles by a factor of $|a|$ in recursions with distinct roots.*

$$|y_{SL}(n)|_{max} < \frac{1}{|a|}|y(n)|_{max} \textit{ for all even-valued } M$$

Proof

Consider the amplitude bound for the SL case, given by (7.47), when M is even-valued. Compare this finite series containing $\frac{M}{2}$ terms with (7.23), the bound without SL in the distinct root case. All terms in (7.47) are contained in (7.23) which has double the number of terms as (7.47). Therefore, for all even-valued M, $|y_{SL}(n)|_{max} < \frac{1}{|a|}|y(n)|_{max}$.

$\square$

SL Reduces Amplitude Bound

Given that the period of the limit cycle is not predictable, the worst-case bound must be used. Theorems 2 and 3 show that the asymptotic values are the worst-case values of the bounds for any period. Theorem 1 establishes the relationship between the amplitude bounds for the asymptotic values with and without SL. Therefore, applying SL to a asymptotically stable second-order recursion with $|a| \geq 0.74$ and with distinct roots *reduces the amplitude bound of the zero-input limit cycle by* γ. The shaded region in Figure 7.1 shows where SL reduces the bound for useful filters.

The result implies that on-line modules with increased precision to contain limit cycles in filters without SL can also be used with SL, without further increasing precision to contain the limit cycle. The disadvantage in the conventional case is that the implementations with and without SL are different, regardless of limit cycle considerations.

7.4 Theoretical Bounds and Simulation Results

This section compares the amplitude bounds of limit cycles derived in the previous section with simulation results. Limit cycle amplitude bounds for two filters, with and without SL, are calculated and plotted for varying periods. Two on-line modules, MA and composite, are used to obtain actual limit cycle amplitudes for the same filters. As expected, the actual limit cycle amplitudes are within the theoretical bounds. The effect of limit cycles on the frequency response of F1 and F2 is shown in Appendix B.

Comparison of Theoretical Bounds

For the distinct root case, the theoretical bound with SL, given by (7.30), is compared against the theoretical bound without SL given by (7.23). The values of the two expressions are computed for periods from 1 to 100 for two filters. The filters are defined as follows.

F1 Filter with coefficients $a = -1.09$ and $b = 0.90$.

F2 Filter with coefficients $a = -1.98$ and $b = 0.99$.

The coefficients of F1 were chosen to correspond to an example in [LnTr73]. Coefficients of F2 were chosen to produce large limit cycles. The limit cycles worsen as $|a|$ and b approach 2 and -1 respectively.

Figure 7.2 shows the amplitude bound for F1 obtained from (7.23) and (7.30). All theorems can be illustrated using the plot. The asymptotic nature of the bound is evident as the period exceeds about 80. The asymptotic bound with SL is 0.46 times the bound without SL, which is within Theorem 1's prediction of 0.58. The asymptotic value is also the highest value of the bound as Theorems 2 and 3 suggest. For all even-valued periods, the bound for SL is lower than the bound without SL as predicted by Theorem 4. The minimum ratio of the bounds for the data set plotted is 0.49, which is within the 0.92 predicted by Theorem 4. Also note that for certain odd-valued periods, 61 and 63 for example, the bound with SL is higher than the bound without SL. For design purposes, the asymptotic value must be used since the period of the limit cycle is not known.

Figure 7.3 shows the amplitude bound for F2 obtained from (7.23) and (7.30). Since the coefficients are extreme, the limit cycle amplitudes are much higher than for F1 and asymptotic behavior is not observed till the period exceeds approximately 500. The ratio of the asymptotic values is 0.25, identical to the predicted ratio.

Theoretical Bounds and Simulation Results

The theoretical asymptotic bounds given by (7.23) and (7.30) are compared in Table 7.1 to the normalized limit cycle amplitudes obtained by simulating F1 and F2, using four on-line implementations. The implementations are based on the MA and composite on-line modules. Each module is used in two configurations, with and without SL. Both on-line modules are radix-4, with digit sets of [-2,2] for the MA module and [-3,3] for the composite module.

As expected, the simulation results are within the bounds. The observed limit cycles amplitudes are not always symmetric in terms of positive and negative amplitudes. For example, the MA module filter for F1 has a positive amplitude of 10 and a negative amplitude of 8. In some cases limit cycles are composed of more than one frequency. For example, the composite/SL filter for F2 has two dominant frequencies, a short period of 2 and a longer period of about 50, while the composite filter for F2 oscillates with a perfectly symmetrical period of 2. The discrepancy between the simulated values and the theoretical values is much larger for the sharper filter F2. The discrepancy may be reduced by using a tighter bound than Long and Trick's. Theoretical bounds must be developed for the SL case using the tighter bound. For example, the bound given in [UnAb75] is particularly suited for sharp filters. The frequency response of F1 and F2 is shown in Appendix B.

Table 7.1: Limit Cycle Amplitudes: Theoretical Bounds and Simulation Results

Filter	*On-Line Module*	*Amplitude from Simulation*	*Theoretical Bound*
F1	MA	10	15.5
	MA/SL	4	7.1
	Composite	6	7.8
	Composite/SL	2	3.6
F2	MA	99	1272
	MA/SL	54	321
	Composite	99	636
	Composite/SL	26	161

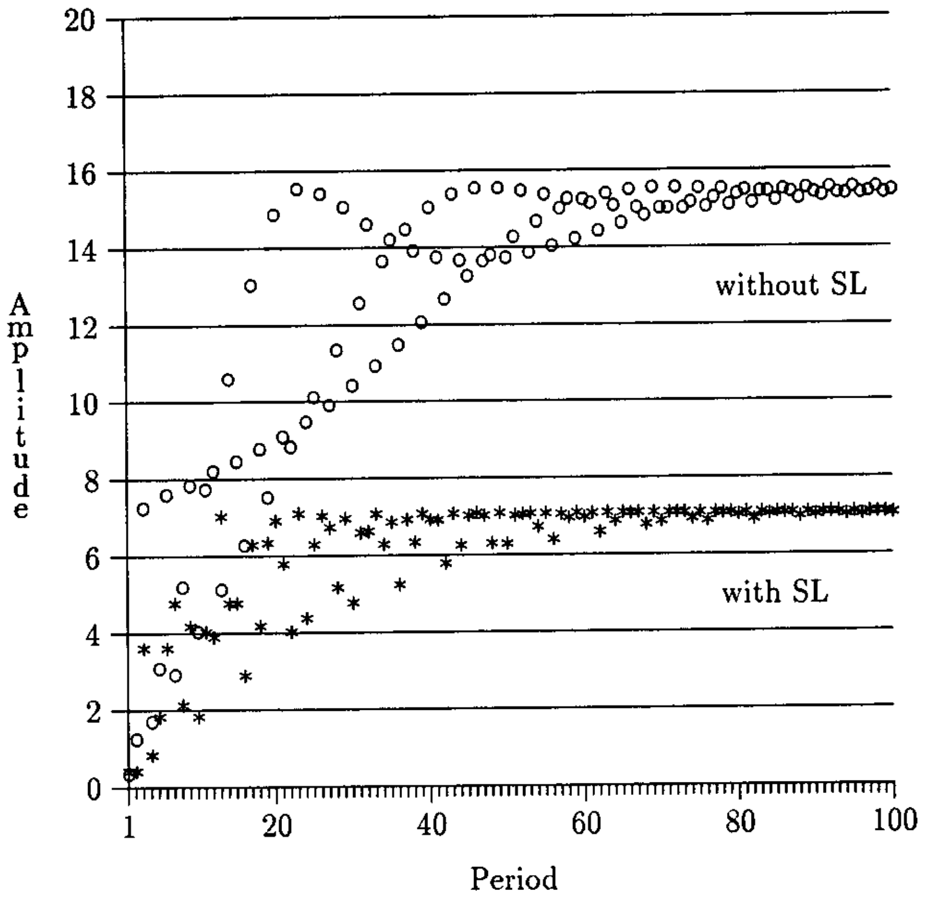


Figure 7.2: Effect of SL on Limit Cycle Amplitude Bounds for F1 ($a = -1.09$, $b = -0.9$)

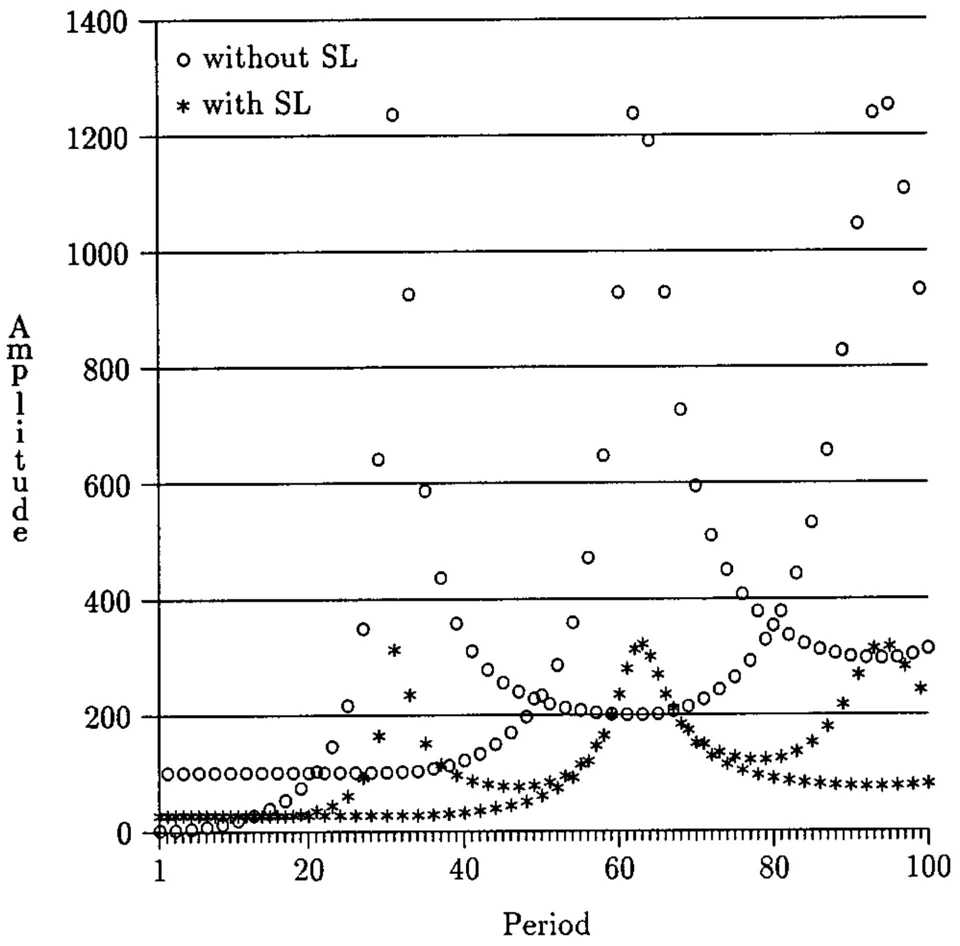


Figure 7.3: Effect of SL on Limit Cycle Amplitude Bounds for F2 ($a = -1.98$, $b = -0.99$)

CHAPTER 8

Dynamic Scaling Scheme for Eliminating Oscillations

The direct form filter structure, unmatched in speed, suffers from inherent non-linear oscillations [Sam88]. An implementation of the structure that eliminates such oscillations without compromising speed is highly desirable, especially when the added cost is small. An on-line multiply-add (MA) module that eliminates all oscillations by extending working precision is described in [Bra89a, Bra89b]. The method is simple, but requires a working precision about 2.5 times the desired precision. Unlike in conventional arithmetic, the clock period in on-line arithmetic is independent of the precision. Thus, this solution preserves the maximum sampling rate of the implementation. However, the increase in cost is significant.

This chapter describes the *Dynamic Scaling* (DS) scheme, a far more attractive solution than simple extension of precision. It is based on an internal floating point representation, adapted to the recurrence expressed by (1.2), while the I/O remains fixed point. The DS scheme is similar to the Block Floating Point scheme for conventional arithmetic used to improve the signal/noise ratio [Opp70]. However, the DS scheme is proposed primarily to eliminate oscillations in a second-order direct form section in a cost effective manner. Section 8.1 describes the basis for the DS scheme, Section 8.2 describes the two basic scaling operations, Section 8.3 describes the DS algorithm, Section 8.4 describes the implementation, and Section 8.5 discusses cost and performance results for a second-order implementation based on two MA modules shown in Figure 4.2.

8.1 Basis for DS Scheme

The DS scheme can be described briefly as follows. With a suitable extension of working precision, the output $y(n)$ can be guaranteed to have a zero value for the most significant digit during zero-input limit cycle oscillations. This allows the output to be left-shifted by 1 digit for the next iteration provided an exponent is introduced to track the shifts. By induction, the shifting can be done until the exponent is decremented to the point at which the output is zero for a given precision. Overflow of $y(n)$ is accomodated by incrementing the exponent and right-shifting the output. Thus, for a given precision, all overflow oscillations and limit-cycle oscillations can be eliminated from the output. Previous results necessary to derive the working precision required for the DS scheme are reviewed next.

Previous Results

The method of extending working precision to eliminate oscillations is described by Brackert [Bra89b]. He derives the number of bits required to eliminate overflow oscillations and limit cycle oscillations. The same notation is used here and is specified next.

m	Number of *fraction bits* used to represent coefficients
b_D	Number of *desired* output bits
b_L	Number of additional least significant bits required to eliminate *limit cycles* from the desired output
b_O	Number of additional most significant bits required to eliminate *overflow* oscillations from the desired output
b_W	*Working* precision in bits
E	Maximum normalized quantization *error* due to multiplication
LC_{max}	Maximum *limit cycle* amplitude

Using the limit cycle amplitude bound derived in [UnAb75], Brackert shows that the limit cycle amplitude bound is given by 8.1.

$$LC_{max} = 2E\left(\frac{4}{\pi}\right)2^{1.5m} \tag{8.1}$$

Using (8.1) and $E = 0.5$, due to rounding, the following equations were derived [Bra89b].

$$\begin{aligned}
b_L &= 1 + \left\lceil 1.5m + \log_2\left(\frac{8E}{\pi}\right)\right\rceil && (8.2)\\
b_O &= 1 + \lceil 1.5m + .5\rceil && (8.3)\\
b_W &= b_O + b_D + b_L && (8.4)
\end{aligned}$$

Note that equations (8.1), (8.2) and (8.3) are for second-order functions with an FIR term (numerator of the transfer function) given by $u(n) = x(n)$. The original equations, (6.3), (6.5) and (6.6) in [Bra89b], are for second-order functions with an FIR term given by $u(n) = x(n) + dx(n-1) + ex(n-2)$. The maximum magnitude of the latter FIR term is 4. Therefore, the modifications that produce (8.2), (8.3), and (8.4) from the original equations are: reduction of LC_{max} by a factor of 4, and subtracting 2 from b_L and b_O. Equation (8.3) assumes fractional inputs. Equation (8.4) gives the total number of bits required to eliminate oscillations by simple increase of working precision as illustrated below.

$$\underbrace{\overbrace{Y\,Y\cdots Y\,Y}^{b_O}\overbrace{Y\,Y\cdots Y\,Y}^{b_D}\overbrace{Y\,Y\cdots Y\,Y}^{b_L}}_{b_W}$$

Precision Required For DS

The dynamic scaling scheme requires the leading digit to be zero for worst-case zero-input limit cycle oscillations. The requirement, for radix-4 digits, is illustrated below, where b_S is the minimum necessary working precision in bits for the DS scheme.

$$\underbrace{0\;0\;\overbrace{Y\,Y\cdots Y\,Y}^{b_L}}_{b_S}$$

Therefore,

$$b_S = b_L + 2 \tag{8.5}$$

and the number of bits needed in addition to b_D, the desired number of I/O bits, is $b_L + 2 - b_D$. Given m, b_S can be calculated from (8.2) and (8.5). The number of bits required for the DS scheme and the simple precision extension scheme for various values of m is shown in Table 8.1. The number of bits required for the precision extension scheme is calculated assuming $b_D = m$.

The exponent used in the DS scheme, E_y, must be large enough to accomodate overflow. The range of E_y, sufficient to accomodate overflow and eliminate limit cycles from the output, is given by (8.6) and (8.7) for radix-4 digits. Thus the DS scheme is capable of guaranteeing an output free of overflow and limit cycle oscillations.

$$E_{max} = \left\lceil \frac{b_O}{2} \right\rceil \tag{8.6}$$

$$E_{min} = -\left\lceil \frac{b_D}{2} \right\rceil \tag{8.7}$$

The working precision for the precision extension scheme is about 2.5 times that required for the DS scheme as Table 8.1 shows. Cost savings result, as shown later in this chapter, because the cost of introducing an exponent into the computation is less than that for precision extension according to (8.4).

Table 8.1: Precision (bits) Required for DS and Precision Extension Schemes

Coeff. frac. bits (m)	8	10	12	14	16	18	20	22	24	26
Bits for DS (b_S)	16	19	22	25	28	31	34	37	40	43
Bits for Prec. Ext. (b_W)	36	44	52	60	68	76	84	92	100	108

8.2 Scaling Operations

The dynamic scaling algorithm is based on two scaling operations, *advance* and *retard*. The advance operation performs a 1-digit left shift of the mantissa and also decrements the exponent. The retard operation performs an n-digit right shift and increments the exponent by n. Table 8.2 illustrates these operations performed on operand y with digits Y.

Figure 8.1 shows the data path of the scaler for executing the scaling operations on an on-line input of 8 digits. The scaler operates as follows. It is assumed that the most significant digit, Y_0, is at the digit input in clock-0. If $n = 0$ and $ADV = 0$ the digits take the *normal* path through the first digit register (at left of Figure 8.1), via the 4-input multiplexer and the 2-input multiplexer, and to the output digit register. The delay for the normal path is two clocks. If $ADV = 1$, the digit stream is *advanced*, because the delay through the scaler is reduced by 1. The retard operation can be described by considering the cases $n < 4$, $4 \leq n < 8$, and $n = 8$, where n is the number of digits retarded. For $n < 4$, selecting inputs to the 4-input multiplexer based on the two least significant bits of n accomplishes the required delay. For example, if $n = 3$ the digit input selected is the output of the last digit register in the cascade of four. The controller that controls the scaler asserts $Z = 0$ for clocks 1,2 and 3, to insert three zero digits. The controller also asserts $R = 0$.

Now consider the case $4 \leq n < 8$. Rather than extend the structure to a cascade of 8 digit registers with an 8-input multiplexer, a cascade of 4 digit registers is used with feedback as shown. A cascade of 4 digit registers is sufficient because the 4 LSDs can be discarded when $n \geq 4$. Immediately after clock-4, the 4 MSDs remain stored in the digit registers. The required delay can be achieved by asserting $R = 1$ in clock-4. Thus the most significant digits are fedback and selected similar to the $n < 4$ case. To introduce n leading zero digits, $Z = 0$ from clock-0 to clock-n. For $n = 8$, $Z = 0$ is asserted for all clocks.

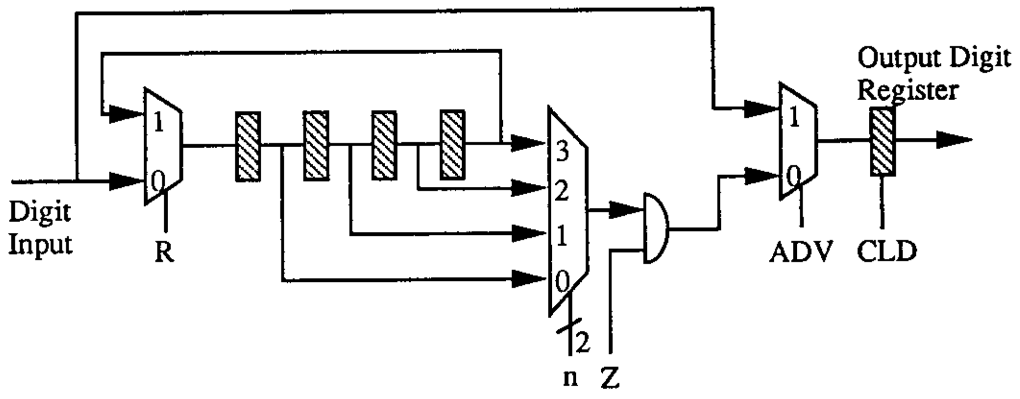


Figure 8.1: Scaler Data Path for 8-digit On-line Operands

Table 8.2: Scaling Operations: Advance and Retard

Operation	Input		Output	
	mantissa	exponent	mantissa	exponent
Advance	$0\,Y\,Y \cdots Y\,Y$	E_y	$Y\,Y \cdots Y\,Y\,0$	$E_y - 1$
Retard 2	$Y\,Y \cdots Y$	E_y	$0\,0\,Y \cdots Y$	$E_y + 2$

8.3 The Dynamic Scaling Algorithm

The DS algorithm is a low-cost floating point scheme designed for the on-line computation of the recurrence specified by (1.2). The DS algorithm is based on the scaling operations defined in the previous section. The variables used in specifying the algorithm are introduced next.

u	normalized $u(n)$
y, y_1, y_2	$y(n)$, $y(n-1)$, $y(n-2)$
E_u, E_y	Exponent of u, Exponent of y_1 and y_2
E'_y	Exponent of y and new exponent of y_1
ADV	$ADV = 1$ signals advance operation
R_u	Magnitude of shift, in digits, for retard on u
R_y	Magnitude of shift, in digits, for retard on y_1 and y_2
R_v	Magnitude of shift, in digits, caused by overflow in y_1 from previous computation; $R_v \in \{0, 2\}$
$Y1z$	$Y1z = 1$ if leading fraction digit of y_1 is 0
$Y2z$	$Y2z = 1$ if leading fraction digit of y_2 is 0

DS Algorithm

```
Begin
Step 0:  Initially $E_y = 0$, $y_1 = 0$, $y_2 = 0$
Step 1:  Compute retard value for $y_1$ and $y_2$
         $R_y = max(E_u - E_y, R_v);\ R_y \geq 0$
Step 2:  Find $ADV$ (boolean value)
         $ADV = (R_v = 0)(Y1z)(Y2z)(E_u < E_y)(E_y > E_{min})$
Step 3:  Compute retard value for $u$
         $R_u = \begin{cases} E_y - E_u - 1 & \text{if } ADV = 1 \\ E_y - E_u + R_v & \text{if } ADV = 0 \text{ and } R_v \geq E_u - E_y \\ 0 & \text{otherwise} \end{cases}$
Step 4:  Compute new exponent
         $E'_y = E_y + R_y - ADV$
Step 5:  Retard/Advance $u$, $y_1$, $y_2$
Step 6:  Execute on-line fixed point computation
         $y = u + ay_1 + by_2$
Step 7:  $E_y \leftarrow E'_y$
Step 8:  Go to Step 1
End
```

Step 1 of the DS algorithm indicates the two conditions that cause y_1 and y_2 to be retarded. One condition is the increase of E_u relative to E_y, and the other is overflow of y_1, the result of the previous iteration. The conditions may occur simultaneously, requiring the maximum retard value to be chosen. Step 2 specifies the conditions for advance: i.e., no overflow, leading fraction digits of u, y_1 and y_2 must be zero, and the exponent must be greater than the minimum value.

As Step 3 indicates, the retard value of u is different because u is normalized. Normalization is convenient because u needs no advance subsequently. Also, advance by more than 1 adds to complexity and delay of the DS unit. Step 5 scales the on-line operands which are input to the on-line fixed point computation in Step 6.

8.4 Implementation of DS scheme

Figure 8.2 shows the block diagram of the DS scheme. Three scalers and a single exponent unit are connected to a module that computes the on-line fixed point recurrence $y = u + ay_1 + by_2$. The block diagram of the scaler was discussed in Section 8.2. The scalers are slightly different. The scaler for u performs only retard operations since u is already normalized. The scalers for y_1 and y_2 produce $Y1z$ and $Y2z$, indicating whether the leading fraction digits are zero or not. The scaler for y_1 also detects overflow and signals the condition with R_v. Overflow can affect one or two integer digits. In either case, the retard value is 2, allowing R_v to be represented by one bit.

The exponent unit computes the retard values and advance signal as shown by the graph in Figure 8.3. Signal timing is shown in Figure 8.4. For 8-digit I/O, the computation cycle consists of 12 clocks, $C0, C1, \ldots, C11$. The exponent unit begins computation in $C10$, when inputs E_y, E_u, R_v, $Y1z$ and $Y2z$ are gated by GATE. Since ADV, R_y and R_u have to be stable in $C1$, the first subtractor shown in Figure 8.3 uses carry select for speed-up. Computation of E_y' is not critical because the value is not needed for the current iteration. In $C0$, the most significant fraction digits are available at the inputs of the scalers shown in Figure 8.2. Since the scalers have a delay of 2 clocks, the scaled outputs are available in $C2$. The signal CLD clears the digit registers of the scalers. CLZ clears the flip-flop that outputs Z in the scalers. The delays shown in Figure 8.2 are for fixed point computation by cascaded MA modules.

8.5 Performance and Cost

This section discusses the sampling rate and cost of the DS scheme implemented in LCA10000 technology. The implementation is compared to the precision extension method for eliminating oscillations. Two examples are used to illustrate the elimination of limit cycles. Fixed point computation is performed by two cascaded MA modules with $r = 4$ and $\rho = 2$.

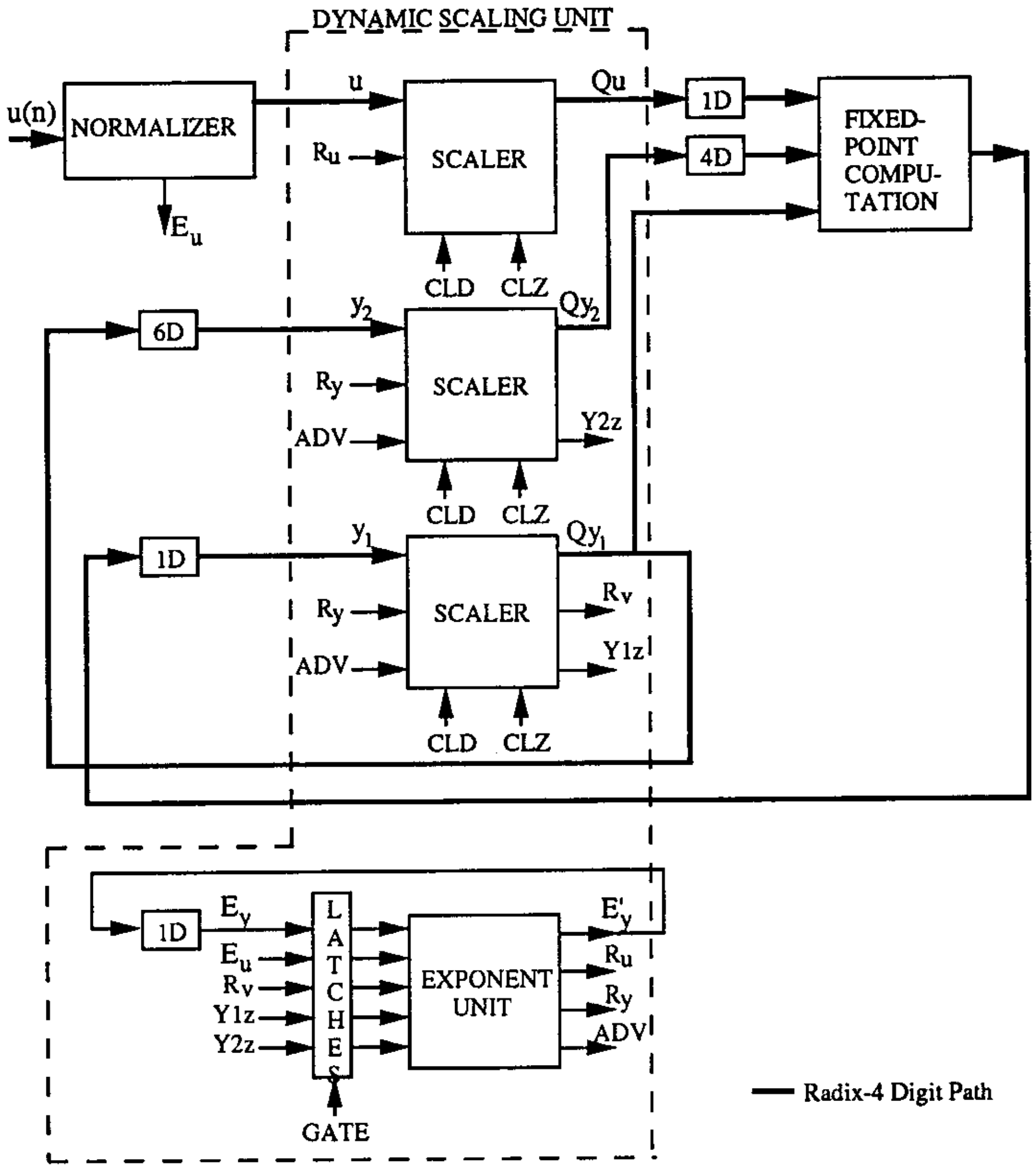


Figure 8.2: Block Diagram of DS Implementation

Rate and Cost of DS Implementation

Table 8.3 describes the clock rates of the components of the DS implementation for I/O words of 8 radix-4 digits (16-bits) with $E_{min} = -15$ and $E_{max} = 15$. A large range for the exponent was chosen to show that the clock rate of the DS Unit is not critical even for large values of m and b_D. As the table shows, the MA modules are the slowest components. The clock rate of the whole design is slower because the wirelength delays, based on the size of the whole design, are larger. The gate count of 4437 for the DS scheme includes the digit delays shown in Figure 8.2.

Table 8.3: Performance/Cost for DS Scheme (16-bit)

Component	Gates	t_{clk} (ns)
DS Unit	1179	9.4
Normalizer	426	<9.4
Cascaded MAs	2424	10
Whole DS Scheme	4437	11.2

Rate/Cost Comparison of Schemes

To compare the DS scheme requires finding the working precision b_W of the *equivalent* fixed point scheme. Table 8.1 shows that for $m = 8$ the required precision for the DS scheme is 16 bits. Selecting $b_D = 8$, and using equations (8.2), (8.3) and (8.4), the working precision for the fixed point scheme described in [Bra89b] is given by $b_W = 14 + 8 + 14 = 36$. The exponent range required for the DS scheme is $-4 \leq E_y \leq 7$. Thus the DS scheme must be compared to a fixed point scheme with 36-bit working precision. The two implementations are compared in Table 8.4. The clock rates are approximately equal because the clock rates of on-line implementations do not change with precision. The fixed point scheme is 16% larger than the DS scheme. Yet, the rate/gate ratio for the DS scheme is twice that for the fixed point scheme.

The cost of a maximum rate word module array with DS can be estimated as follows. First consider the DS scheme for 16-bit I/O. The fixed point computation requires 4 word modules in an array similar to Figure 5.3. Only one DS unit is required. The DS unit is placed in front of one word module and overflow in the other three word modules is accomodated by a 3-digit extension of precision. The normalizer is required to perform block normalization of 4 independent inputs $u(n)$ through $u(n+3)$. Block normalization produces 4 on-line inputs with a single exponent E_u. Block normalization is equivalent to performing 4 separate normalization operations, selecting the maximum exponent as E_u, and retarding the other on-line inputs appropriately. The cost of the array with DS is estimated as follows.

Block Normalizer	$\simeq$1,000
8 MA modules (22-bit I/O)	11,856
DS Unit	1,179
Array with DS	$\simeq$14,000

With fixed point precision extension, each word module requires a precision of 36 bits. This requires 6 word modules, or, 12 MA modules. The number of gates required is 28,260 (= 6 $\times$ 4710). Thus, the fixed point scheme is twice as large as the DS scheme for maximum rate arrays. The theoretical maximum rate of the DS scheme may be slightly less than the rate of the fixed point array, due to the 2-clock latency of the DS unit. The maximum rate for an array without DS is given by $MaxRate = \frac{1000}{t_{clk}\delta_{imp}}$. With DS, the maximum rate is given by $MaxRate_{DS} = \frac{1000}{t_{clk}(\delta_{imp}+\frac{2}{N_{DS}})}$, where $N_{DS} = \left\lceil \frac{\delta_{imp}+d+d_{ovf}}{\delta_{imp}} \right\rceil$. The additional precision required to absorb overflow in the word modules without a DS unit is d_{ovf}. For the example considered, $d = 8$, $d_{ovf} = 3$, $\delta = 2$, and $\delta_{imp} = 4$. Thus $MaxRate_{DS}$ is 88% of $MaxRate$, assuming the clock period is identical for both

Table 8.4: DS Scheme vs. Fixed Point Scheme

Scheme	Gates	Rate (Msamples/s)	t_{clk} (ns)	Rate/Gate (Msamples/s/gate)
DS Scheme	4437	7.44	11.2ns	0.00167
Fixed Point	5130	4.13	11.0ns	0.00081

arrays. Overall, the array with DS has a rate/cost ratio almost twice that of the fixed point array.

Comparison of Limit Cycles

Both the DS scheme and the fixed point precision extension scheme eliminate limit cycles from the desired output (without affecting the sampling rate). This is illustrated in Figures 8.5 and 8.6, for two 2-pole filters F1 and F2. The DS scheme is compared to a 22-bit fixed point scheme. The 22-bit scheme is equivalent to the 36-bit scheme without the 14-bit precision extension for overflow oscillations. A 16-bit fixed point scheme is also included for reference. The plots are obtained by plotting $\log_2(amplitude)$ against the sample number of the output in response to an initial impulse of 0.0625. As the plots show, the 16-bit fixed point scheme suffers from zero input limit cycles. Also, the response of the DS scheme dies out a little earlier than the 22-bit fixed point scheme. Thus the scheme performs slightly better than the fixed point scheme. This is expected since the full precision of the DS scheme is used until the exponent is reduced to a minimum. In contrast, the number of non-zero digits reduces with time in the fixed point scheme.

Conclusion

The Dynamic Scaling scheme eliminates limit cycles oscillations and overflow oscillations in on-line implementations of direct form structures. The two main components of the scheme are the DS unit and the module for performing fixed point computations. For a DS scheme using one word module, the sampling rate is about 80% higher than for the fixed point precision extension alternative. In addition, the DS scheme is about 13% smaller. For maximum rate arrays, the DS scheme is slightly slower but only half the size of the alternative. In both cases rate/cost ratio is doubled. In addition to eliminating oscillations, the DS scheme eliminates the need for scaling between second-order sections in cascade.

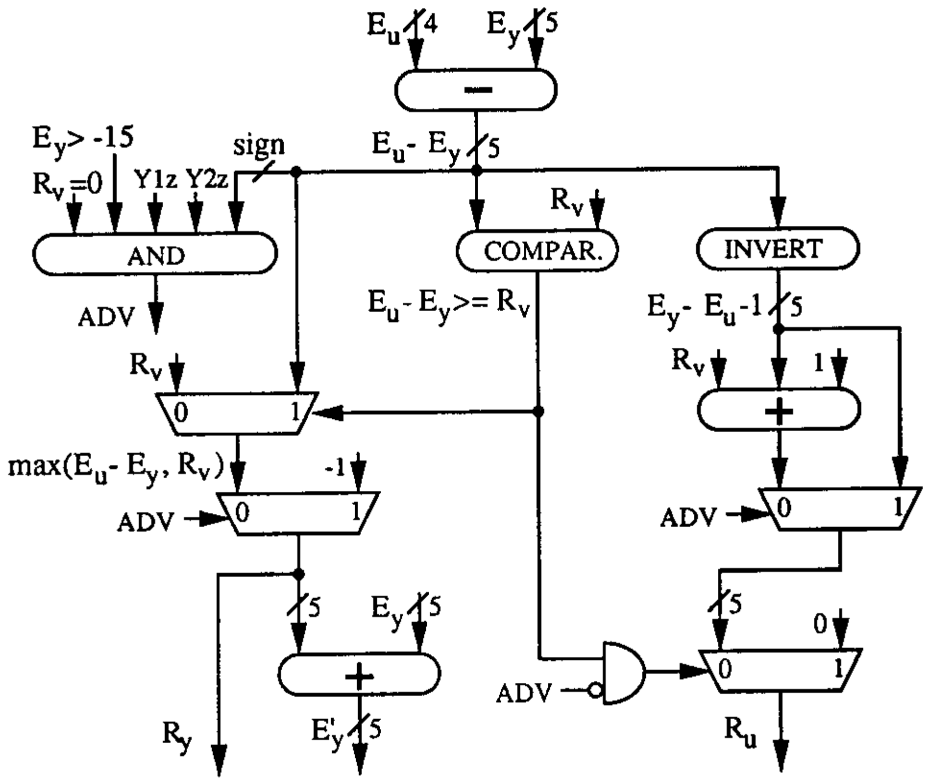


Figure 8.3: Exponent Unit Computations

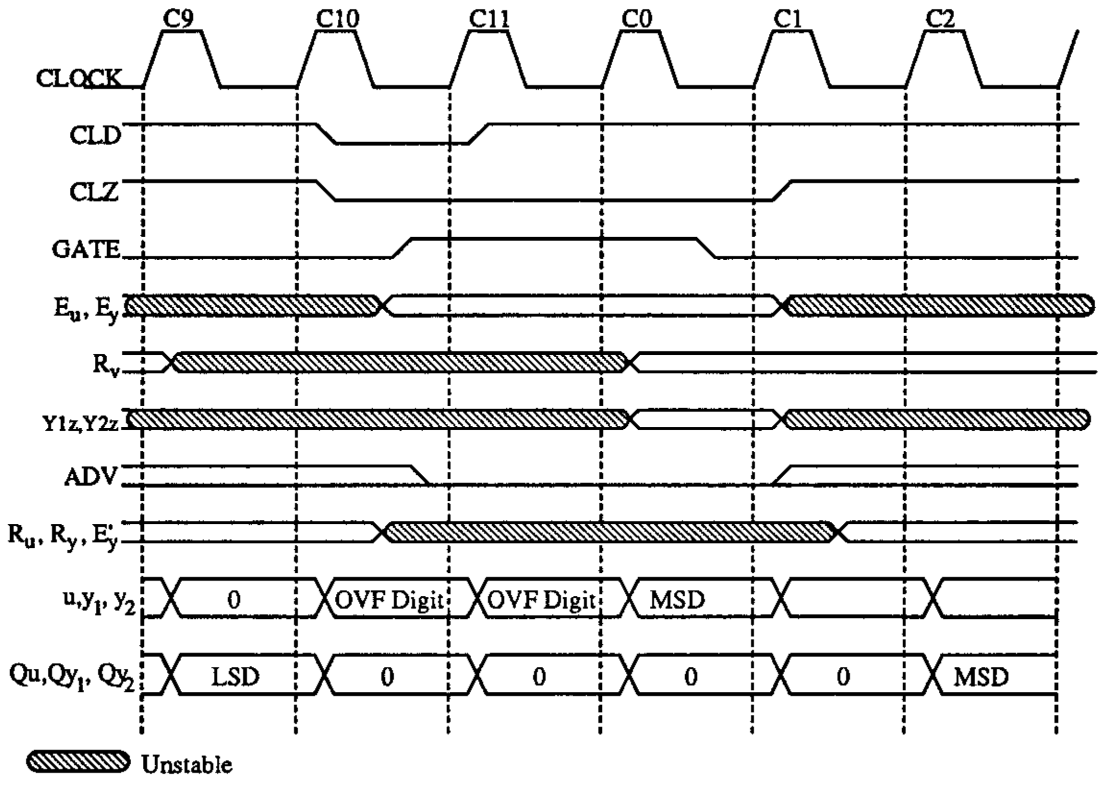


Figure 8.4: Timing for DS Scheme

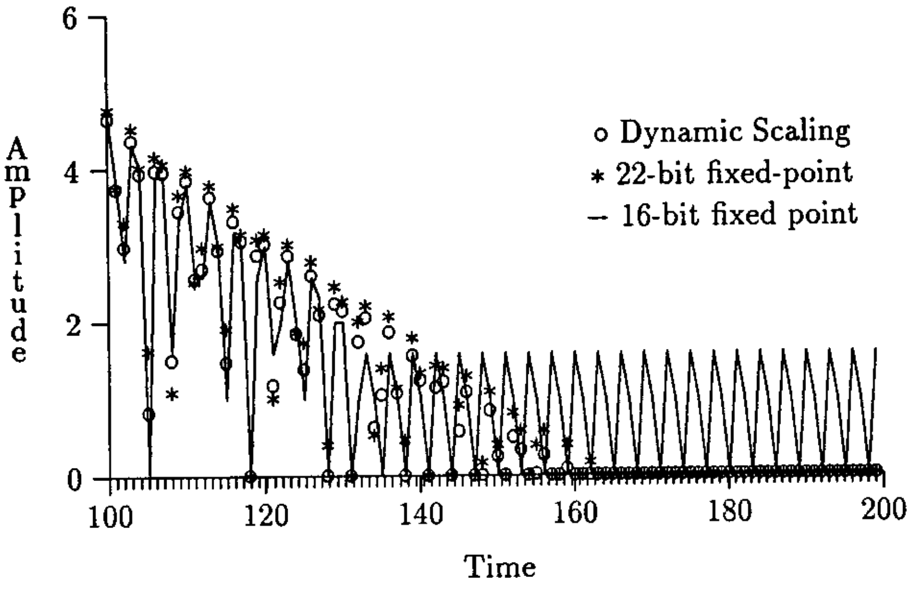


Figure 8.5: Amplitude Response for F1 (a = −1.09, b = −0.9), amplitude= log_2(16-bit output)

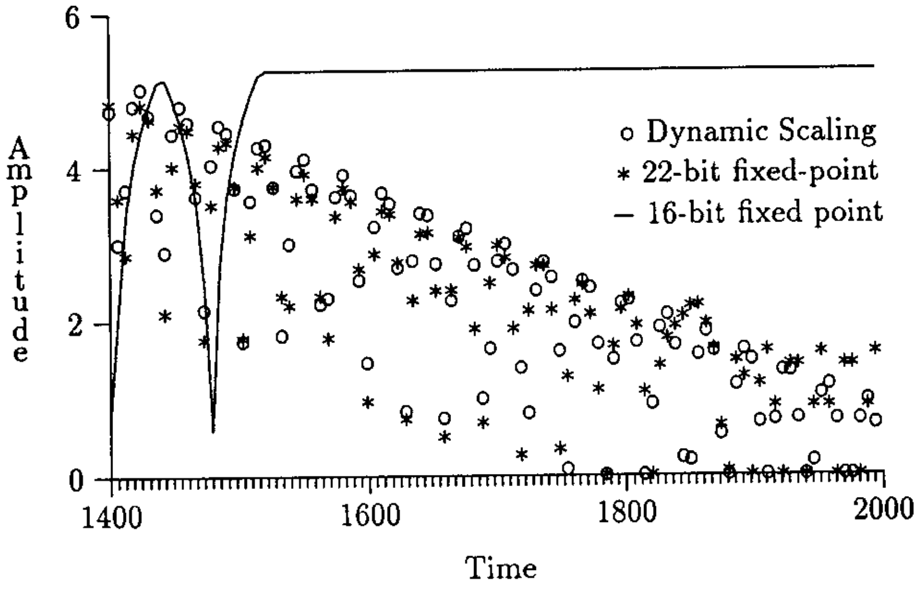


Figure 8.6: Amplitude Response for F2 (a = −1.98, b = −0.99), amplitude= log_2(16-bit output)

CHAPTER 9

Evaluation

The previous chapters discussed on-line implementations that compute the second-order recursions given by (1.1) or (1.2). Three types of basic modules were analyzed: simple schemes composed of separate on-line units, composite modules, and schemes based on MA modules. Using these basic modules, word module arrays and digit stage arrays were developed to achieve maximum sampling rate. In Chapter 6, two lookahead transformations were applied to achieve higher rates with arrays derived from the same basic modules. The performance and cost of these on-line arrays were compared to a conventional implementation with 2 levels of SL. Chapter 8 described the DS scheme, a cost-effective method of eliminating non-linear oscillations in on-line designs. The scheme uses the basic word modules without modification.

The first section of this chapter evaluates the most cost-effective implementations, including lookahead schemes. Section 9.2 reviews previous implementations described in recent publications, and Section 9.3 reviews solutions to the problem of non-linear oscillations.

9.1 Evaluation of Implementations

The basic word modules were developed by choosing the radix and digit set that minimized the on-line delay of the digit recurrence algorithm. For LCA10000 technology [LSI87], the radix-4 designs with $\delta = 2$ were shown to provide the highest performance for composite modules and MA modules. The implementations have two or three stages with $\delta_{imp} = 4$. Word module arrays and digit stage arrays were derived from basic composite modules and MA modules. To facilitate discussion, the arrays are named with acronyms which are deciphered as follows:

Array name: < *basic_module* > [< *array_type* >< ρ >]/[< *SL_level* >]

basic_module	Conv	conventional
	MA	on-line multiply-add module
	S2C	on-line 2-stage composite module
	S3C	on-line 3-stage composite module

array_type	WM	word module array
	DS	digit stage array

ρ	2 or 3	digit set of basic module is $\{-\rho, \ldots, \rho\}$
SL_level	SL1 or SL2	Level of SL used if any

For example, MAWM2 is an on-line word module array composed of MA modules with digit set of $\{-2, \ldots, 2\}$. Figure 9.1 shows the performance and cost of several arrays. The graph plots the maximum rate against the number of gates required per array for $d = 8$. The bottom of the plot shows three arrays, S2CWM3, S3CWM3 and MAWM2, that compute expression (1.1) without lookahead. The array with the lowest rate and fewest gates is S2CWM3, an array of three composite word modules with $\rho = 3$. The 3-stage composite word module is used in S3CWM3 to achieve a faster rate at lower cost than S2CWM3. The fastest and most modular array is MAWM2, composed of 9 MA word modules with $\rho = 2$.

Scattered lookahead, applied to the transfer function corresponding to (1.1), eliminates the dependency of $y(n)$ on $y(n-1)$ and doubles the sampling rate. The IIR part of the transformed expression, given by (1.2), is computed by an array of word modules or digit stages. The FIR part of the expression, $u(n)$, can be computed easily, using an on-line or conventional approach, and is not discussed here. The gate count of composite modules computing expression (1.2) is estimated from the gate count of the corresponding composite module computing (1.1). Clock rates of corresponding composite modules are the same since the critical paths are identical. Figure 9.1 shows three arrays, S3CWM3/SL1, S3CDS3/SL1 and MAWM2/SL1, computing the IIR part for one level of scattered lookahead. S3CWM3/SL1 is an array of 3-stage composite word modules with $\rho = 3$. S3CDS3/SL1 is the corresponding digit stage array with the same rate but requiring fewer gates than S3CWM3/SL1. MAWM2/SL1 is a word module array composed of 12 MA word modules with $\rho = 2$. It is the fastest and most modular of the arrays for one level of scattered lookahead.

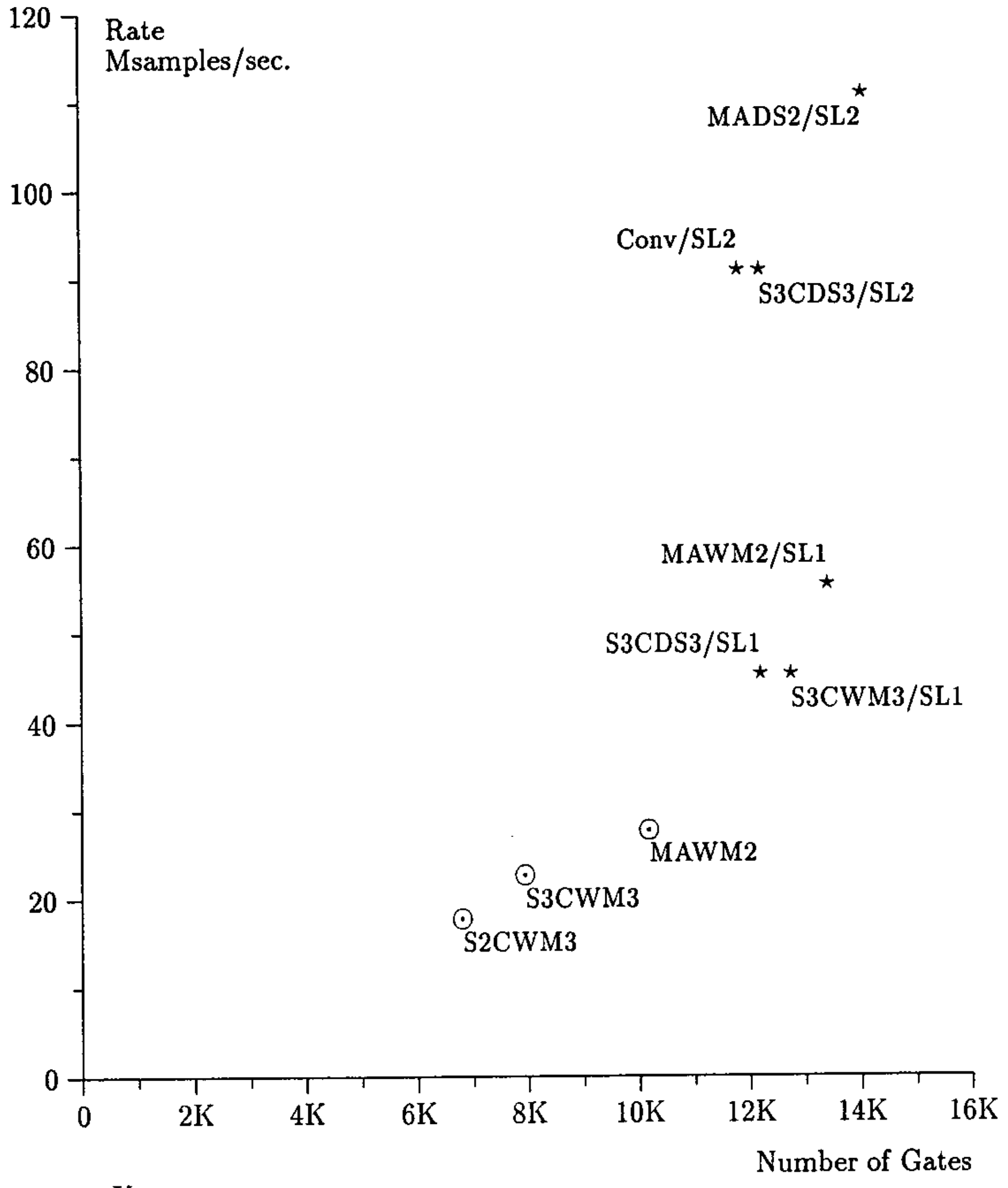
For two levels of scattered lookahead, digit stage arrays are more cost effective than word module arrays. Two on-line schemes, MADS2/SL2 and S3CDS3/SL2, and one conventional implementation, Conv/SL2, are shown in Figure 9.1. Array S3CDS3/SL2 is the same array as S3CDS3/SL1. The higher rate is obtained by doubling the utilization of the array. MADS2/SL2 is a cascade of two digit stage arrays based on the MA module with $\rho = 2$. Conv/SL2 is a conventional implementation for two levels of scattered lookahead. Radix-4 recoding is used for the multiplier. The design has the same precision of $d = 8$ as the on-line schemes.

The DS scheme eliminates all non-linear oscillations without compromising the sampling rate, except in a maximum rate array of word modules. In such an array, the sampling rate achievable is slightly less than maximum. The advantage of the DS scheme is that the cost of realizing a given sampling rate is about half that of using Brackert's precision extension scheme [Bra89b].

9.2 Previous Implementations

Several implementations of IIR filters have been reported in recent papers. This section compares the performance and cost of these implementations which fall into three categories. The first category consists of schemes using conventional multiply-add units (LSB first, bit-parallel). The second category consists of systolic arrays using signed-digit representations to allow most-significant-digit-first operation. Input and output words in such arrays are in skew-parallel form. The third category is the on-line or residual recurrence schemes investigated in this dissertation, and in [Bra89a] and [Bra89b].

The implementations referred here give the number of transistors required as the cost measure. To compare these designs to those developed in this dissertation, critical paths and sampling periods are expressed in terms of gate delays, and the number of gates are converted to the equivalent number of transistors. Based on [LSI86], the number of transistors to a gate is taken to be four (i.e., two P transistors and two N transistors are equivalent to one gate). The criteria for comparing designs were described in Section 3.4.



Key

S2: 2-stage, S3: 3-stage
⋆ Equation (1.1), ⊙ Equation (1.2)

Figure 9.1: Comparison of On-line and Conventional Implementations ($d = 8$)

Conventional Schemes

Conventional designs use multiply-add units which compute from LSB to MSB with carry-propagation. The inputs and outputs are in bit-parallel form. The main disadvantages of such designs are:

- Slower than on-line or MSDF designs, especially with increasing precision. In conventional designs, the recursive part of the computation cannot be pipelined unless lookahead is used. With one level of scattered lookahead the critical multiplication can be pipelined into two stages, and with two levels of scattered lookahead four stages can be used [PrMs87a].
- Techniques for limiting nonlinear oscillations introduce delays in the recursive part of the loop [MtLw81].
- Speed reduces as word precision increases.

Lookahead techniques increase the rate of conventional and on-line implementations. In general, lookahead techniques introduce the possibility of imperfect pole-zero cancellation in the passband of the filter. No evidence of this was found in the case of two filters examined in Appendix B. Nevertheless, the frequency response of a filter using SL can differ significantly from the ideal in some cases [BlCh91].

A high speed implementation of a fourth-order filter, composed of two cascaded second-order sections, using two levels of scattered lookahead is presented in [PrHt88]. The details of their implementation are given in Table 9.1. The implementation illustrates the scattered lookahead and decomposition techniques proposed by Parhi et al. [PrMs87a].

Table 9.1: Parhi and Hatamian Design

Sampling Rate	*Transistors*	*Technology*	*Coefficient Range*	*Precision (bits)*
100MHz	77,600	0.9μ (AT&T)	[-4,4]	10

This is the fastest conventional design reported and consists of 17 multiply-add units. The multiply-add units used in this design are based on regular carry-save arrays with carry-ripple addition. No recoding is used. The design is sensitive to increase in word length because of carry ripple. The main purpose of the design is to demonstrate the high-speed achieved by SL. Non-linear effects due to scattered lookahead are not considered.

The conventional design described in this dissertation cannot be compared with the Parhi and Hatamian design due to differences in technology and precision. The conventional design presented in Appendix A reduces the product into carry-save form using trees of full adders. The carry-save product is recoded into a radix-4 signed-digit representation, before being used as the multiplier in the next iteration. Since recoding takes constant time, and carry propagation time increases with precision, the speed of this multiplier is less sensitive to increase in word length. The disadvantages are the irregular nature of the reduction trees and the larger number of gates.

MSDF and On-line Arrays

Most-significant-digit-first (MSDF) and on-line arrays use signed-digit arithmetic in combination with regular carry-save representations. First-order radix-2 IIR filters with MSDF systolic array architecture are described by Knowles, McCanny, McNally and Woods [KnMc88], [KnMc90]. A simple extension of this scheme to radix-4 is described in [LFH90]. However, first-order sections are not useful in practice [Bra89b]. The development of the MSDF systolic arrays in [KnMc88], [KnMc89], [KnMc90], [LFH90] and [KnMc91] is not as systematic as the MSDF and on-line arithmetic approaches described in [ErLn89]. The latest implementation by McNally et al. is a 40MSample/sec second-order IIR filter chip using the MSDF systolic array architecture developed for the first-order filters [KnMc91]. The features of this architecture are described next.

The systolic array designs have the output, $y(n)$, and the recursive input, $y(n-1)$, in skew-parallel form, where each digit is delayed by one clock and produced or consumed MSD first. These designs are very similar to the digit-stage scheme presented in [ErLn89]. The difference is that the FIR term is fed as an initial value in bit-parallel form in the MSDF systolic arrays. This requires deskewing and converting the output to 2's complement representation to conform to the input form required for the next section in the cascade. In the digit stage schemes presented in this dissertation, the FIR term is in on-line form. Table 9.2 summarizes the filter described in [KnMc91].

The filter is packaged in a 84-pin chip and represents a complete design with scan paths for testability. The 1-clock latency ($\delta_{imp} = 1$) of this design is achieved by using 1 level of SL and combining three digit stages into one. Thus the radix-2 signed digits are produced in groups of three. The design includes saturation logic to handle overflows and a scaling constant for each second-order section. The saturation logic mitigates the effects of overflow by clamping the output to ± 1. The hardware overhead is small and the critical path is extended by two gate delays [MnMc90]. Limited scaling is achieved by shifting the FIR input of each second-order section by 0, 1, 2 or 3 digits, providing scale factors of 1, 0.5, 0.25 and 0.125 [KnMc91]. Further scaling may reduce the S/N ratio.

The DS scheme by comparison, provides a complete solution to the limit cycle and overflow problems, while improving the S/N ratio with dynamic scaling. Of course, the hardware cost is higher than for implementing saturation logic and limited scaling. In addition, the DS scheme does not require changes in the fixed-point modules as the MSDF array does.

Table 9.2: Design by McNally et al.

Sampling Rate	*Transistors*	*Technology*	*Digit Set*	*Coefficient Range*	*Precision (bits)*
40MHz	60,000	1.5μ	[-1,1]	2	12

On-Line MA Modules

A 3-stage design of an on-line MA module is described by Brackert in [Bra89a] and [Bra89b]. The design described in this dissertation has two stages. Using four on-line MA modules, identical in design to Brackert's, a second-order direct-form II filter with programmable coefficients was implemented by Cha [Cha91]. The first two rows of Table 9.3 summarize the implementations by Brackert and Cha respectively. The third row summarizes the implementation described in Chapter 4. The higher rate of the design described in Chapter 4 is due to the fewer number of stages and the faster technology. Timing models used by the simulators may also differ.

Table 9.3: Comparison of On-Line MA Modules

Sampling Rate	*Transistors*	*Technology*	*Digit Set*	*Coeff. Range*	*Precision (bits)*
7.14MHz	4778	1.5μ (VTI)	[-2,2]	2	16
5.88MHz	-	2μ (MOSIS)	[-2,2]	2	-
27.7MHz	4732	1.5μ (LSI)	[-2,2]	2	16
32.7MHz	4732	0.7μ (LSI)	[-2,2]	2	16

The last row of Table 9.3 shows the performance and cost of a MA module with 16-bit I/O implemented in a LCA200K[1] gate array [FnEr92b]. The architecture of the MA module is identical to that described in Chapter 4. The timing shown in Table 9.3 for LCA200K is for a worst-case commercial environment. With 2 levels of SL, this MA module can be used to achieve a maximum sampling rate of 128MSamples/sec.

To compare the designs developed here with others implemented in different technologies, critical paths and sampling periods are expressed in terms of gate delays. For the MA module described here, $\delta_{imp} = 4$ and the critical path is approximately 12 gate delays. Thus the sampling period of this MA module without SL is 48. The radix-4 on-line MA module described Brackert et al. is much slower: $\delta_{imp} = 5$ and the critical path is approximately 14 gate delays [Bra89a]. The main reason for the difference is that their design has one more stage than the MA module described here. For the McNally et al. design, the critical path is 28 gate delays and the latency is two clocks without SL. Thus the sampling period is 56 gate delays, slightly greater than that for the module described here.

Conventional Designs

The conventional design described in Appendix A can be compared to the implementation by Parhi and Hatamian [PrHt88]. Their 100MHz design demonstrated the scattered lookahead method described in [PrMs87a]. The design is a fourth-order filter, with 2 levels of SL for 10-bit I/O and 15-bit working precision,

[1]LCA200K is 0.7μ HCMOS gate array [LSI92].

implemented in a 0.9μ custom CMOS technology. The critical path is about 24 gate delays. The multiplication in their design uses a ripple carry adder in the critical loop. This produces a more regular design, compromising speed at longer wordlengths. The conventional design described in Appendix A allows 16-bit I/O words, uses radix-4 recoding and eliminates the CPA from the critical loop. It has a critical path of about 16 gate delays, two thirds of the Parhi and Hatamian implementation.

9.3 Non-Linear Oscillations: Bounds and Solutions

Chapter 7 derives new absolute bounds for zero-input limit cycle oscillations that occur in recursions transformed by SL. Four theorems are proven to show that the worst-case limit cycle amplitude decreases for filters using one level of SL with coefficients in the region shown in Figure 7.1. The coefficients of sharp filters would be well within this region. Thus, a filter that eliminates limit cycles from its output without SL will do so with SL as well.

The dynamic scaling scheme described in Chapter 8 eliminates all non-linear oscillations from its output. The scheme is twice as cost-effective as Brackert's precision extension scheme. The DS scheme provides the high sampling rate of the direct form structure without its inherent oscillations. In addition, the DS scheme improves the signal/noise ratio because it performs automatic scaling similar to a floating point scheme.

CHAPTER 10

Conclusion and Further Research

This chapter concludes the dissertation, summarizing the main contributions and suggesting ways of extending this research. The research analyzes VLSI implementations for computing second-order recursions using on-line arithmetic.

The main contributions of this dissertation are:

1. Analysis of the design space of on-line implementations, especially for gate arrays, for computing second-order linear recursions. The best designs were implemented as basic modules.

2. Design of maximum rate arrays using basic modules.

3. A fast conventional implementation for second-order linear recursions with SL.

4. Derivation of absolute bounds for zero-input limit cycles in recursions transformed by SL.

5. Design and implementation of the DS scheme: a new algorithm for cost-effective elimination of non-linear oscillations and improving S/N ratio.

As a result of 1 and 2 above, the appropriate on-line architecture for a given sampling rate is known. Also known, due to 3, is the *cross-over word length*, i.e., the word length at which on-line implementations become faster than conventional ones. The results depend on the technology. The results and their applicability to other technologies is discussed later.

Optimal designs were obtained by selecting radix and digit set that maximized sampling rate for implementations in LCA10000 technology [LSI87]. An I/O word length of 16 bits was chosen for comparison with a conventional implementation. The architecture that provides the fastest rates for this technology has a 2-stage or 3-stage pipeline with $\delta_{imp} = 4$, producing one sample every four clocks.

The following conclusions may be drawn from the analysis of the designs and implementations in LCA technology.

1. With no lookahead, word module arrays provide the same performance at a lower cost, compared to digit stage arrays. With one level of scattered lookahead, digit stage arrays become as cost-efficient as word module arrays. With two levels of lookahead, digit-stage arrays are more cost-effective.

2. The arrays based on MA modules with $\rho = 2$ provide the highest performance, with or without lookahead.

3. The digit stage array based on the MA module with $\rho = 2$ is faster than the conventional implementation computing the IIR part of the recursion with 2 levels of scattered lookahead for 16 bit I/O words. Thus, the cross-over point between on-line and conventional is about 12 or 14 bits for LCA10000 technology.

Applicability of Results to Other Technologies

The actual performance and cost will be different for other technologies. The change in the relative positions of the designs in Figure 9.1 depends on how different the relative speeds and relative costs of similar cells are. The relative positions will also depend on the variety and level of cells that are available to the designer. For LCA10000 technology, the cells provided are at the level of full adders, half adders, and multiplexers [LSI87].

The relative clock rates of the designs depend on the relative delays of the critical paths. For the on-line architectures considered, the relative delays of the critical paths may not change much. For example, in any technology, a 3-to-2 reduction is faster than a 5-to-2 reduction. Thus, an MA module will be faster than a 3-stage composite module, and the latter will be faster than a 2-stage composite module. Similarly, a smaller digit set will permit faster multiplication regardless of technology.

It is difficult to compare the MA digit stage array and the conventional implementation in another technology using simple critical path analysis. This is especially so for large designs, where regularity is crucial for fast clocking and compact layout. Therefore, the cross-over point between on-line and conventional implementations in a different technology may occur at a different word length.

The relevant cost measure in most technologies is the area, which depends on the number of transistors *and* the regularity of the design. For example, consider the array of MA word modules with one level of a scattered lookahead, and the array of 3-stage composite word modules. In a custom design, the greater regularity of the array of MA word modules would very probably result in widening the performance gap and closing the cost difference between these two arrays.

A better comparison of performance and costs must include measures other than those considered here. For example, design complexity, power consumption, and number of I/O pins are important cost factors. The performance measures not considered in this dissertation include the ease of eliminating non-linear oscillations, dynamic range and noise. These performance factors are especially important in comparing on-line implementations with conventional ones, and evaluating the usefulness of lookahead methods.

Suggestions for Further Research

Suggestions for future research in developing on-line and conventional arithmetic algorithms for application-specific requirements are outlined in this closing section. Most of the research in this dissertation has focussed on design alternatives for high-speed cost-effective implementations for computing linear recurrences in a gate array technology. Figure 10.1 shows this research in a broader context of computations, optimized characteristics, technology, and applications. Motivated by application, different paths in the figure outline related areas of research.

For example, a FPGA (Field Programmable Gate Array) implementation requires minimum-area algorithms and structures, where speed is compromised for compactness. Bit-serial arithmetic has been shown to produce area-time efficient implementations of DSP algorithms in FPGAs [GrTn92]. With inherently low communication overhead, on-line algorithms are appropriate for such implementations. Minimum-area implementations are also useful where several functions are implemented on a chip. Algorithms that consume minimum power for a given computation rate are useful in several limited-power applications. Such algorithms are also required to reduce operating temperatures in large chips.

As shown here and elsewhere, choosing the optimum algorithm and design for a given application yields appreciable performance/cost benefits. For the problem considered in this research, the optimum design was chosen by individual analysis of several algorithms. It is highly desirable to develop a rapid method for selecting the optimum algorithm.

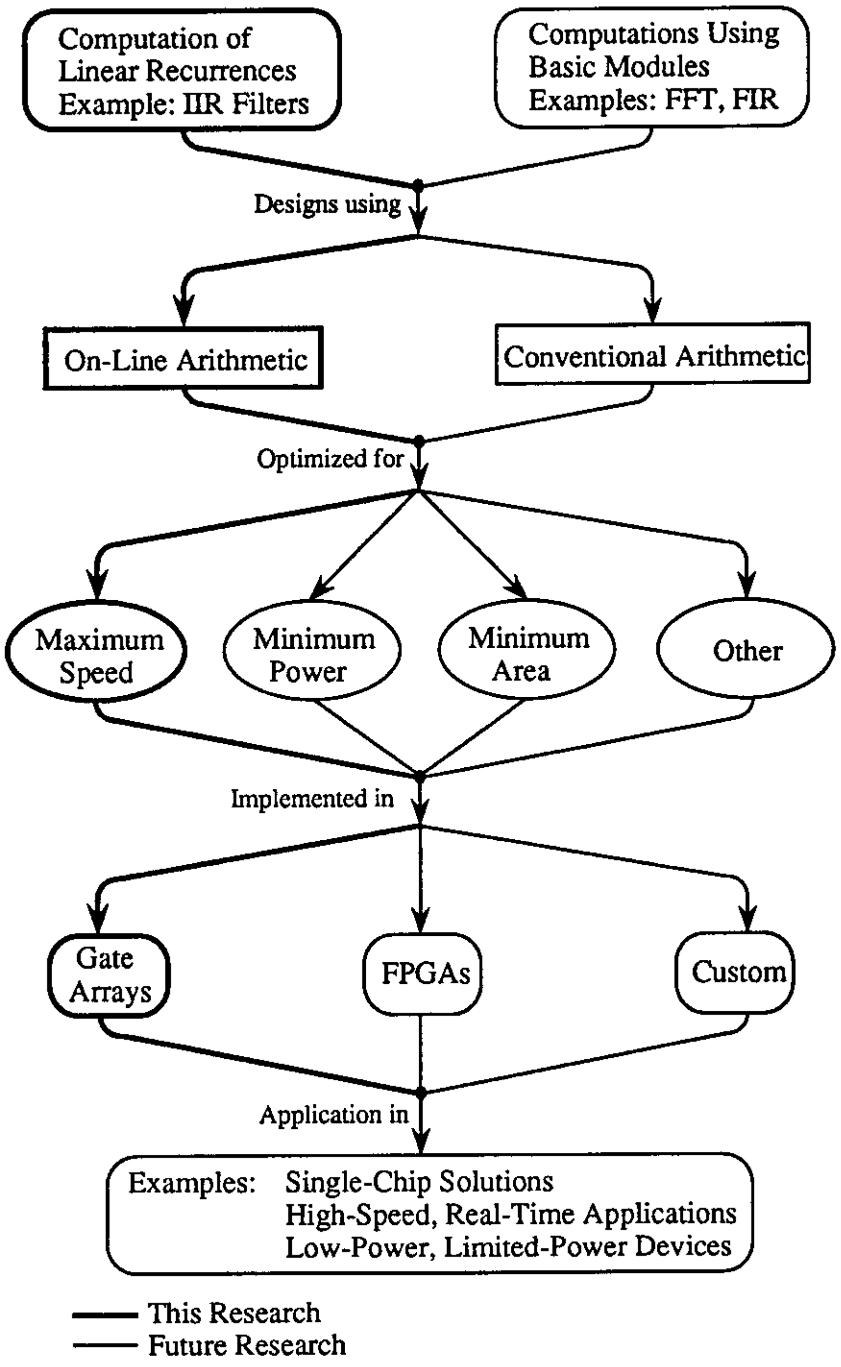

Computation of Linear Recurrences Example: IIR Filters
Computations Using Basic Modules Examples: FFT, FIR
Designs using
On-Line Arithmetic
Conventional Arithmetic
Optimized for
Maximum Speed
Minimum Power
Minimum Area
Other
Implemented in
Gate Arrays
FPGAs
Custom
Application in
Examples: Single-Chip Solutions
High-Speed, Real-Time Applications
Low-Power, Limited-Power Devices
This Research
Future Research

Figure 10.1: Suggestions for Future Research

APPENDIX A

Conventional Design of a Second-Order IIR Section with Scattered Lookahead

The design described here uses conventional arithmetic (bit parallel, least significant bit first) to implement a second-order IIR section with two levels of scattered lookahead. The original implementation by Parhi and Hatamian [PrHt88] cannot serve as a basis for comparing on-line schemes, due to the differences in word length and technology. The 16-bit design described here is implemented in a LCA10000 [LSI87] gate array, the same gate array used for the on-line implementations. The design introduces a method of recoding the multiplier, represented in carry-save form, into higher radix signed digits. The method is suited for conventional high speed recursive multiplication with long word lengths.

Figure A.1a shows the IIR computation, the sampling rate of which is determined by the 4-stage multiply-add. The goal is to minimize the clock period of the 4-stage multiply-add. Section A.1 describes the architecture of the 4-stage multiplier. Section A.2 describes the implementation and Section A.3 summarizes performance and cost of the design.

A.1 Architecture of 4-Stage Multiply-Add

Figure A.1a shows the delays introduced by two levels of scattered lookahead, that allow 4-stage pipelining of the critical inner loop. The outer loop has 8 delays and the inner loop has only 4, making the inner loop critical. The delays can be distributed to allow pipelined multiplication and addition. Figure A.1b shows the distribution used in the implementation described here. The inner loop is a 4-stage multiply-add. The outer loop is a 7-stage multiplier, sharing one stage with the 4-stage multiply-add unit. The 4-stage multiply-add multiplies its input by the coefficient and adds the output of the 7-stage multiplier and the FIR term.

The specifications of the multiply-add unit are as follows.

1. The computation, using generic notation, is : $y(n) = c \cdot y(n-4) + u_1(n) +$

$u_2(n)$, where c is a constant and u_1 and u_2 are the terms to be added.

2. A 4-stage design.

3. Multiplicand and multiplier are 16-bit words. In this design the multiplicand is constant.

4. LCA10000 technology [LSI87].

The factors that affect the clock rate, besides the technology, are the limited number of stages (4) and the number of bits (16) of the multiplier and multiplicand. The difference between this set of specifications and that of the Parhi and Hatamian design is in the technology and the 10-bit precision of the multiplier and multiplicand. An architecture appropriate for a higher precision recursive multiply-add in LCA10000 [LSI87] technology is described next.

Architectural Options

Architectural choices are made at the algorithmic level and at the component/layout level. At the component/layout level, the decision is whether to use a conventional array or Wallace Trees (n-to-2 counters) to reduce the partial products. At the algorithmic level, the following have to be considered.

1. Decide whether the constant is assigned to be the multiplicand or the multiplier.

2. Recoding the multiplier into higher radix signed digits, to produce fewer partial products.

3. Carry-save representation of result to avoid CPA .

The Parhi and Hatamian design does not use recoding [PrHt88]. Speed is achieved at the component/layout level by exploiting the regularity of the conventional array. A carry-ripple adder, taking 2 of the 4 stages, is used to produce the result. As precision increases, this approach suffers increased delays.

The number of partial products can be reduced by recoding the multiplier into higher radix signed digits. Tree structures can also be used to speed up the reduction of partial products. The CPA can be entirely eliminated by using the carry-save representation of the result as described by Kleine and Noll [KlNo88]. In the Kleine and Noll Scheme, the coefficient is recoded into the digit set $\{-2,-1,0,1,2\}$. The advantage of their scheme is that recoding costs nothing. The disadvantage is that the number of partial products remain the same; the number of partial products is doubled by the carry-save representation, and recoding reduces that by half. A larger digit set cannot be used with the Kleine and Noll Scheme because the generation of higher multiples of the variable input is difficult.

The scheme developed here recodes the carry-save representation of the multiplier. Since the multiplicand is constant, multiples of the multiplicand can be pre-stored, allowing the use of larger digit sets and higher radices. The recoded multiplier can be used by both multipliers. With radix-4, the number of partial products is reduced from 16 to 8. The disadvantage of recoding delay is offset by having fewer partial products.

Multiplication Scheme

The multiplicand is represented in 2's complement and the multiplier is taken to be a 2's complement value in carry-save representation. The multiplier, without loss of generality, is considered to be a fraction. The 2's complement multiplication scheme consists of three steps: recoding the carry-save form of the multiplier into a signed-digit representation, producing partial products, and reducing the partial products into carry-save form. The reduction is done in the usual manner with an array of Wallace Trees. The recoding and the partial product generation steps are summarized next.

1. The multiplier is represented by
$$\sum_{i=0}^{m-1} C_i 2^{-i} + \sum_{i=0}^{m-1} S_i 2^{-i} = -X_0 + \sum_{i=1}^{m-1} X_i 2^{-i}$$

2. To recode multiplier, add bits of same weight and produce transfer digits, t_i, and sum digits, s_i:
$$\begin{aligned} t_{m-1} &= 0 && \text{(A.1)} \\ t_{i-1} &= \left\lfloor \frac{C_i + S_i + 1}{2} \right\rfloor,\ t_i \in \{0,1\},\ i = 1,2,\ldots,m-1 && \text{(A.2)} \\ s_0 &= (C_0 + S_0) \bmod 2,\ s_0 \in \{0,1\} && \text{(A.3)} \\ s_i &= C_i + S_i - 2t_{i-1},\ s_i \in \{-1,0\},\ i = 1,2,\ldots,m-1 && \text{(A.4)} \end{aligned}$$

3. Recoded form of multiplier is $\sum_{i=0}^{m-1} D_i 2^{-i}$, where
$$\begin{aligned} D_i &= s_i + t_i,\ D_i \in \{-1,0,1\},\ i = 1,2,\ldots,m-1 && \text{(A.5)} \\ D_0 &= (s_0 + t_0) \bmod 2,\ D_0 \in \{0,1\} && \text{(A.6)} \end{aligned}$$

4. Partial product generation for all digits except the MSD is done in the usual manner: partial product $= 2^{-i}D_i \cdot Multiplicand$. For the most significant digit, the partial product is obtained as follows.

$$\text{partial product} = \begin{cases} 0 & \text{if } D_0 = 0 \\ Multiplicand & \text{if } D_0 = 1 \text{ and } \sum_{i=1}^{n} D_i 2^{-i} < 0 \\ -Multiplicand & \text{otherwise} \end{cases}$$

Higher radix digits are formed by grouping and recoding the corresponding radix-2 digits. The recoded digit set has minimum redundancy.

A.2 Implementation

The first step in implementing the scheme is to decide on the radix for recoding the multiplier. Higher radices incur larger recoding delays and larger partial product generation delays. The advantage of higher radix is that fewer partial products are generated. Table A.1 shows an estimate of the delays (excluding set-up and hold delays) for radices 2,4 and 8. The table also gives the number of FAs and HAs for the array. The total number of gates must also take into account the recoding, partial product generation and registers and latches.

Table A.1: Delays and Costs of Components of 4-Stage Multiply-Add

	Radix-2	Radix-4	Radix-8
Number of FA levels	9	7	5
Number of FAs	231	126	80
Number of HAs	80	67	42
Recode Delay (ns)	2.2	3.1	5.4
Product Gen. Delay (ns)	2.5	4.0	4.4
Total Delay (ns)	24.5	21.5	20.8

The table shows that the radix-2 design is slightly slower and much larger than the other designs. The radix-8 design has a slightly lower total delay than the radix-4 design. However, the radix-8 design has a larger recode delay, requiring an entire stage for the recoder. This leaves only three stages for partial product generation and reduction. The critical stage would then have 3 FAs, with a delay of 6.6ns. The radix-4 design also has a critical stage with 3 FAs. Thus, the two designs have similar speeds. The costs are also similar, because the advantage of fewer product terms is offset by larger recoding logic and larger product generating multiplexers. The radix-4 scheme is chosen because the shorter delays make it easier to balance delays in all stages.

Figure A.2 shows a slice of the 4-stage recursive multiply-add, with the CPA eliminated from the critical loop. The FIR term and the output of the 7-stage multiplier constitute the add inputs. The output of the multiply-add is recoded into radix-4 signed digits. The recoded multiplier is also an input to the 7-stage multiplier. The recoder, the MSD partial product generator, and the array are described next.

A digit slice of the recoder is shown in Figure A.3a. The recoder input is the 16-bit carry-save form. The input is divided into 8 groups. Each group, consisting of adjacent sum bits and the corresponding carry bits, is recoded into one radix-4 digit D_i, $D_i \epsilon \{-2,-1,0,1,2,3\}$ and $D_0 \epsilon \{0,1,2,3\}$. Figure A.3b shows the MSD generator. To produce the partial product of the MSD, the sign of the value represented by the other digits must be determined. Then the sign of the MSD is changed, if necessary, and the partial product is obtained as usual. Due to the delay in finding the MSD, the partial product of the MSD is available only at the third stage of the reduction (Figure A.2).

Each partial product of 18 bits is generated by 18 6-input multiplexers. The data inputs of the multiplexers are fed by coefficient multiples. The pre-stored multiples of the coefficient, c, are: c, $-c$ and $3c$. The 8 partial products and the two add inputs are reduced by the array of FAs and HAs shown in Figure A.4a. To obtain 16-bit precision, an 18-bit carry-save result is produced and the 2 lsb positions are used for rounding. Thus, as Figure A.4a shows, only part of the array is implemented. The full array needs to be implemented only if proper rounding is done, based on the CPA result of the least significant 16 bits. Since a CPA in the recursive loop reduces the sampling rate, approximate rounding is performed based on the 18-bit carry-save result.

The design of the 7-stage multiplier is similar to the 4-stage multipier. A stage is considered to be the network between two registers. The 7-stage multiplier is composed of three stages that generate and reduce the 8 partial products to a carry-save form and a 4-stage CPA that produces a 2's complement 16-bit result. It receives a recoded multiplier as an input from the 4-stage multiply-add unit.

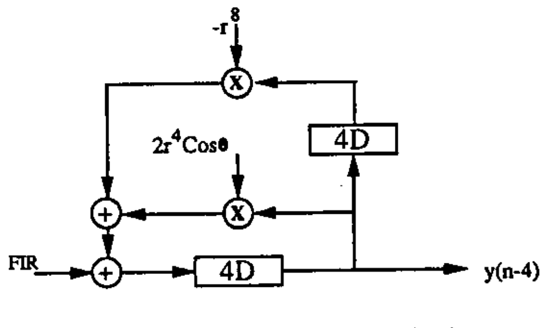
The multiplicand is a constant coefficient. Five levels of FAs and HAs reduce 8 partial products to a carry-save form, as shown in Figure A.4b.

A.3 Performance and Cost

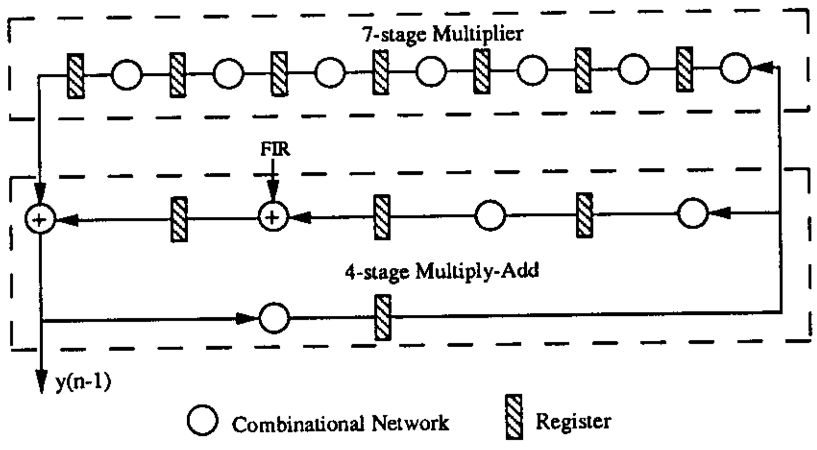
The entire IIR section consists of the 4-stage multiply-add, the 7-stage multiplier and a CPA to form the output of the IIR section. A simple carry ripple adder may be used to add the carry-save result and produce a 2's complement output. However, for convenience, an instance of the 4-stage adder designed for the 7-stage multiplier is used. The gate count and clock period for the IIR section is given in Table A.2 for an LCA10000 [LSI87] implementation.

Table A.2: Performance/Cost for Conventional IIR Section (16 bits) with 2 levels of Scattered Lookahead

Component	*Gates*	t_{clk}
4-stage multiply-add	5345	11ns
7-stage multiplier	5349	11ns
CPA (4-stage)	1102	8ns
Entire IIR section	11796	11ns



(a) Delays Introduced by Scattered Lookahead



(b) Delays Distributed for Pipelining

Figure A.1: IIR Computation with 2 Levels of Scattered Lookahead

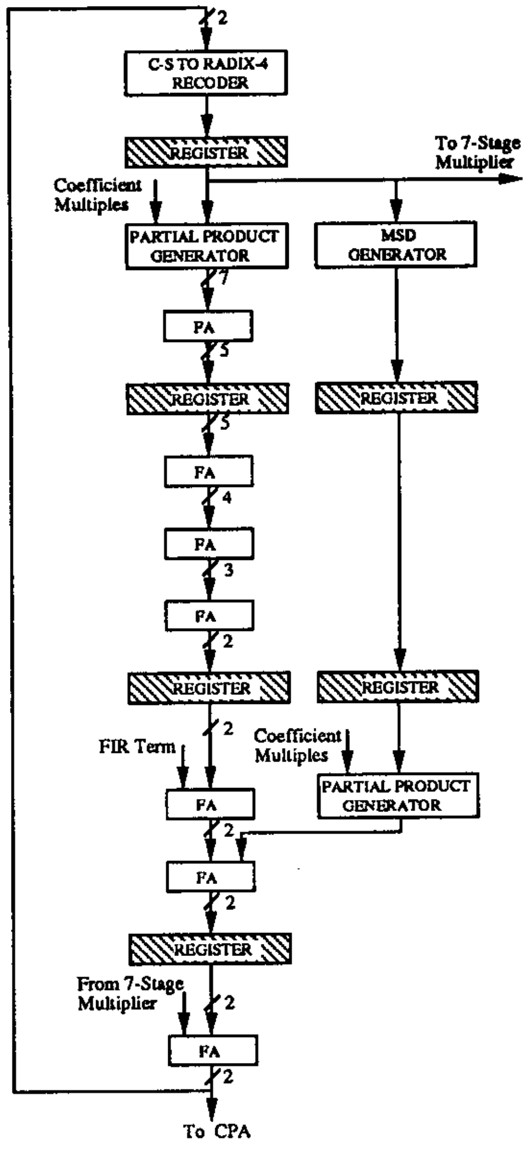
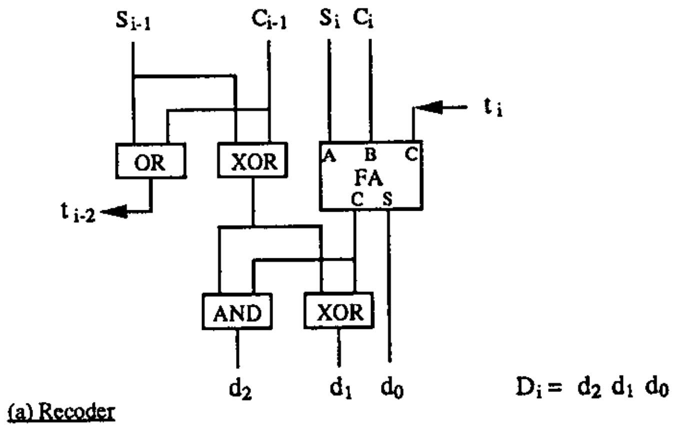


Figure A.2: Slice of 4-Stage Multiply-Add (16 bits)



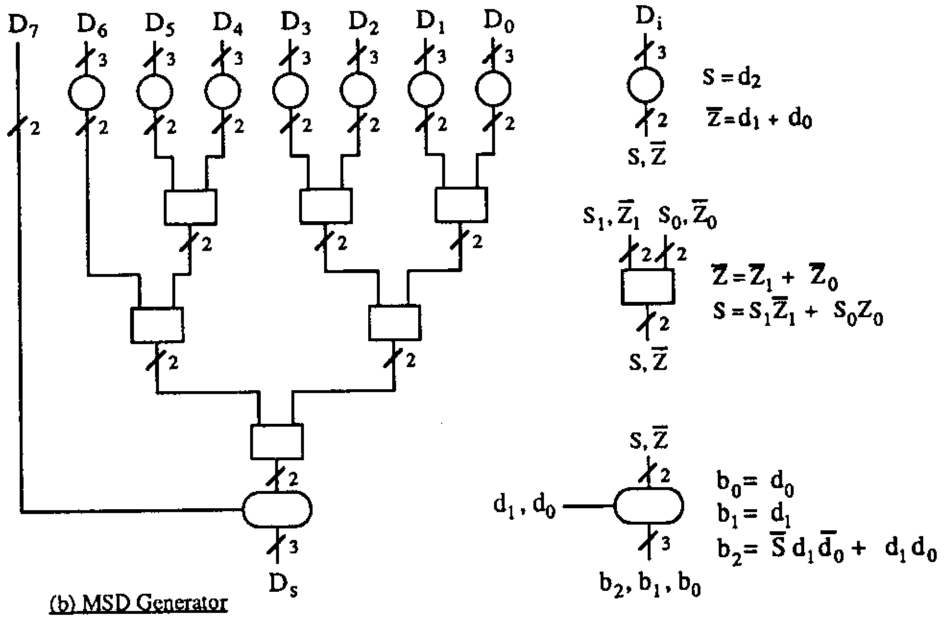
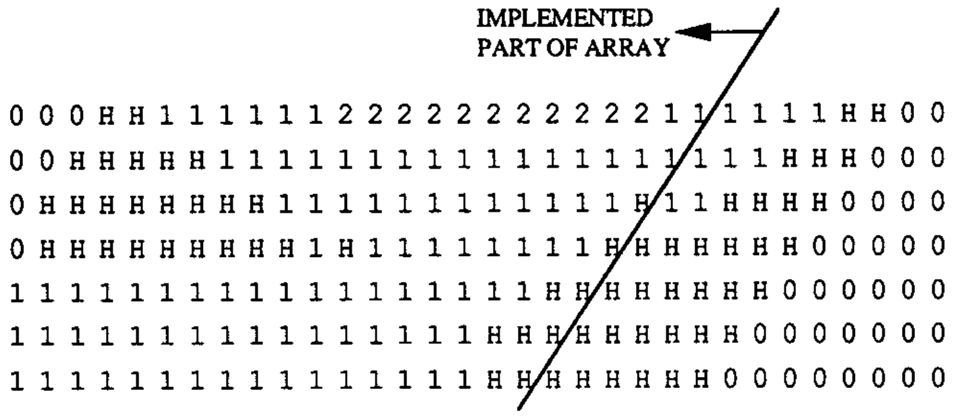
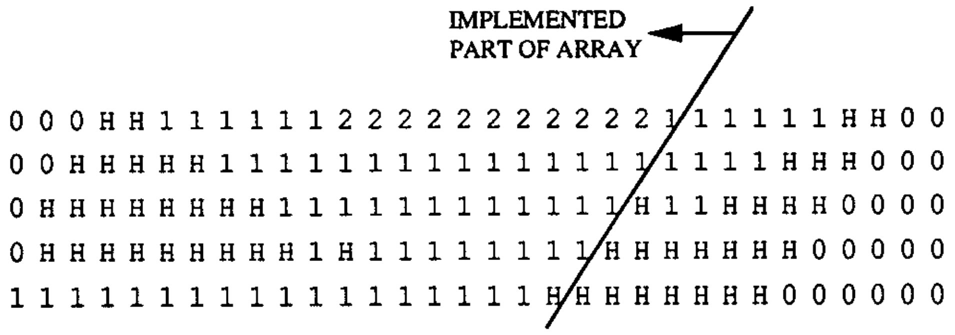

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
D_s
(b) MSD Generator
D_i
$S = d_2$
$\overline{Z} = d_1 + d_0$
$S, \overline{Z}$
$S_1, \overline{Z}_1$ $S_0, \overline{Z}_0$
$\overline{Z} = \overline{Z}_1 + \overline{Z}_0$
$S = S_1\overline{Z}_1 + S_0 Z_0$
$S, \overline{Z}$
$S, \overline{Z}$
d_1, d_0
$b_0 = d_0$
$b_1 = d_1$
$b_2 = \overline{S}\, d_1 \overline{d}_0 + d_1 d_0$
b_2, b_1, b_0

Figure A.3: (a)Carry-Save to Radix-4 Recoder (b) MSD Generator



```
0 0 0 H H 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 H H 0 0
0 0 H H H H H 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 H H H 0 0 0
0 H H H H H H H H 1 1 1 1 1 1 1 1 1 1 1 1 H 1 1 H H H H 0 0 0 0
0 H H H H H H H H H 1 H 1 1 1 1 1 1 1 1 H H H H H H H 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 H H H H H H H H 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 H H H H H H H H H 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 H H H H H H H H 0 0 0 0 0 0 0 0
```

(a) Array of Adders for 4-Stage Multiply-Add



```
0 0 0 H H 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 H H 0 0
0 0 H H H H H 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 H H H 0 0 0
0 H H H H H H H H 1 1 1 1 1 1 1 1 1 1 1 1 H 1 1 H H H H 0 0 0 0
0 H H H H H H H H H 1 H 1 1 1 1 1 1 1 1 H H H H H H H 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 H H H H H H H H 0 0 0 0 0 0
```

(b) Array of Adders for 7-Stage Multiplier

Key:		
	0	No Adder
	H	Half Adder
	1	Single Full Adder
	2	Two Full Adders

Figure A.4: Arrays for Reducing Partial Products

APPENDIX B

Frequency Responses of Filter Examples

This section compares the effects of scattered lookahead (SL) and dynamic scaling on the frequency response of two second-order filters, F1 and F2. These filters serve as examples elsewhere in this work. The expression for a second-order filter with two zeros at the origin and two complex conjugate poles is given by (B.1). Applying one or two levels of SL transforms (B.1) to (B.2) or (B.3).

$$
\begin{aligned}
y(n) &= x(n) + ay(n-1) + by(n-2) && \text{(B.1)}\\
y(n) &= x(n) + ax(n-1) - bx(n-2) \\
&\quad + (a^2+2b)y(n-2) - b^2y(n-4) && \text{(B.2)}\\
y(n) &= x(n) + ax(n-1) + (a^2+b)x(n-2) + a(a^2+2b)x(n-3) \\
&\quad - b(a^2+b)x(n-4) + ab^2x(n-5) - b^3x(n-6) \\
&\quad + (a^4+4a^2b+2b^2)y(n-4) - b^4y(n-8) && \text{(B.3)}
\end{aligned}
$$

The frequency response is obtained by Fourier Transform of the output of the filter response to an impulse of magnitude 0.0625. This magnitude is an arbitrarily chosen value, small enough not to cause overflow. The number of samples in the response is 1024 for F1 and 2048 for F2. The filter model uses MA modules with I/O words of 16 bits.

Figure B.1 shows the responses of F1 with 0,1, and 2 levels of SL. The responses deviate from each other only in the regions A, B, and C. These deviations are due to limit cycle oscillations. The regions are shown on a magnified scale in Figures B.2, B.3 and B.4. The largest deviation due to limit cycles, shown in Figure B.3, occurs when no SL is used. Table B.1 shows the largest percentage deviations with respect to a freqency response obtained by direct double precision computation.

Figure B.5 shows the frequency responses of F2 with 0,1, and 2 levels of SL. The responses are almost identical except in region D, shown magnified in Figure B.6. As in the case for F1, the deviations are due to limit cycle oscillations. As Table B.1 shows, the worst deviation occurs when no SL is used. The reduction is limit cycle oscillations for the filters using SL is in accordance with Theorem 1 of Chapter 7. The best response, shown in Figure B.7, is obtained from the filter using dynamic scaling.

The examples used here show that the frequency response, particularly in the pass band, is not appreciably affected by SL for filters F1 and F2 with 16-bit I/O. The same is true in the case of a elliptic low pass MSDF systolic array filter using 1 level of SL and 12-bit I/O [KnMc91]. The effect of SL on noise gain, relative to the filter without SL, is studied rigorously in [BlCh91]. This paper shows that the noise gain increases near the origin (in z-plane) but depends on the pole angle near the unit circle. Some pole positions near the unit circle tend to increase the noise gain drastically while others decrease it. Overall, the numerical performance of SL is reported to be good [BlCh91].

Table B.1: Deviation of Frequency Response due to SL

Filter	*Level of SL*	*Max. % Deviation*	*Frequency*
F1	0	12.5	0.33496
F1	1	9.5	0.15136
F1	2	8.3	0.83008
F2	0	26.1	0.49975
F2	1	3.8	0.01562
F2	2	8.1	0.01810

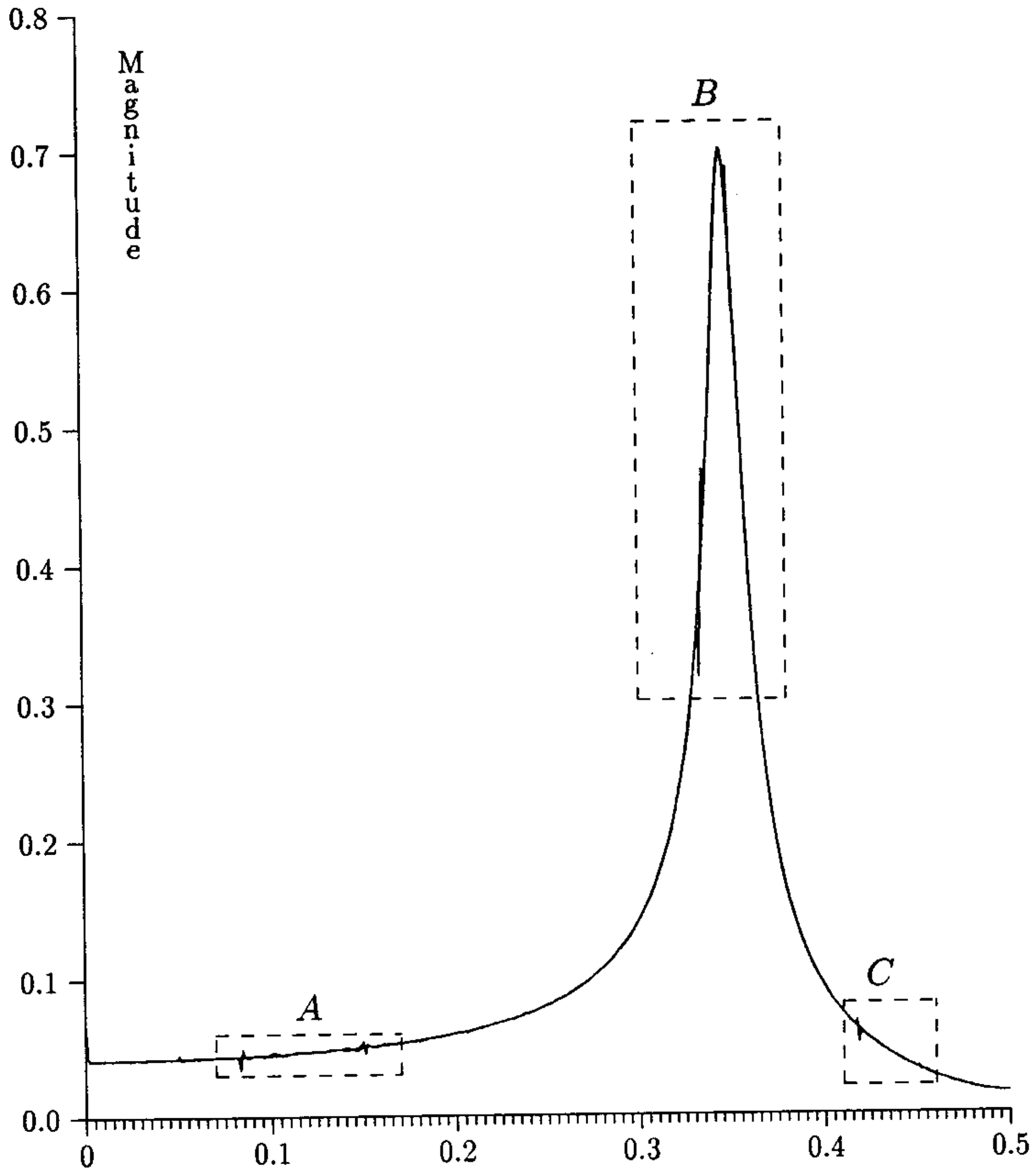


Figure B.1: Frequency Response of F1 ($a = -1.09$, $b = -0.90$)

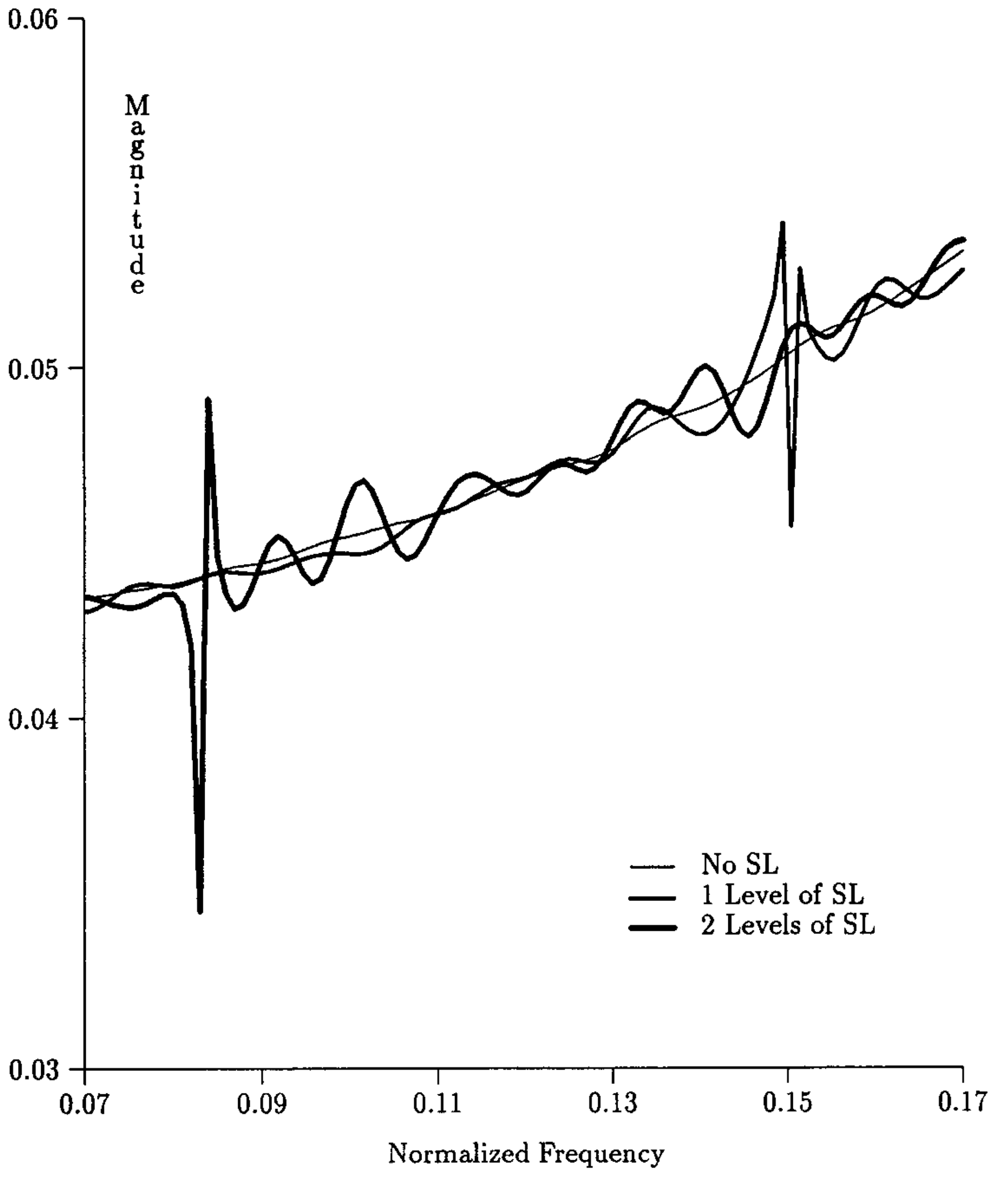


Figure B.2: Region A of Frequency Response of F1 ($a = -1.09$, $b = -0.90$)

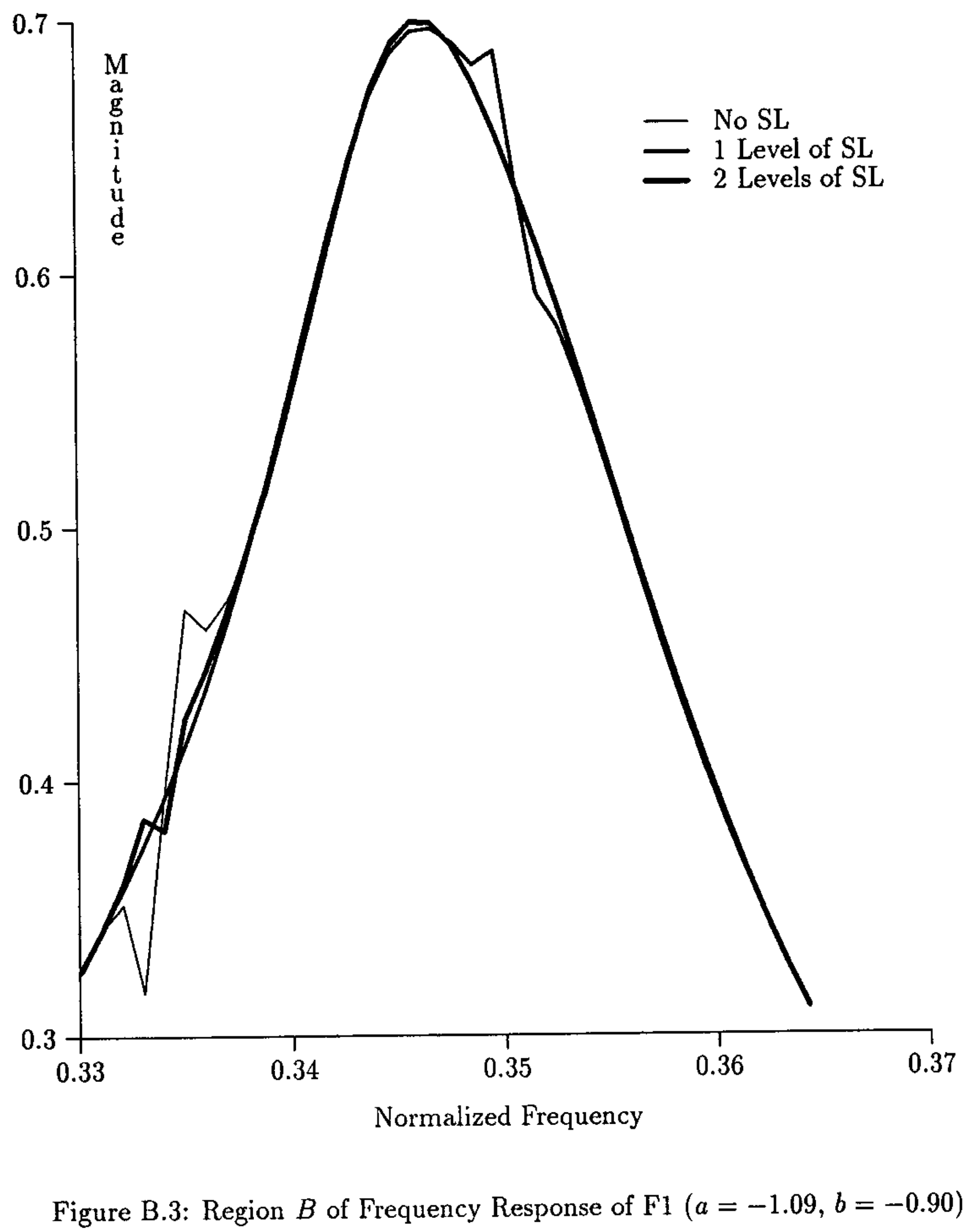


Figure B.3: Region B of Frequency Response of F1 ($a = -1.09$, $b = -0.90$)

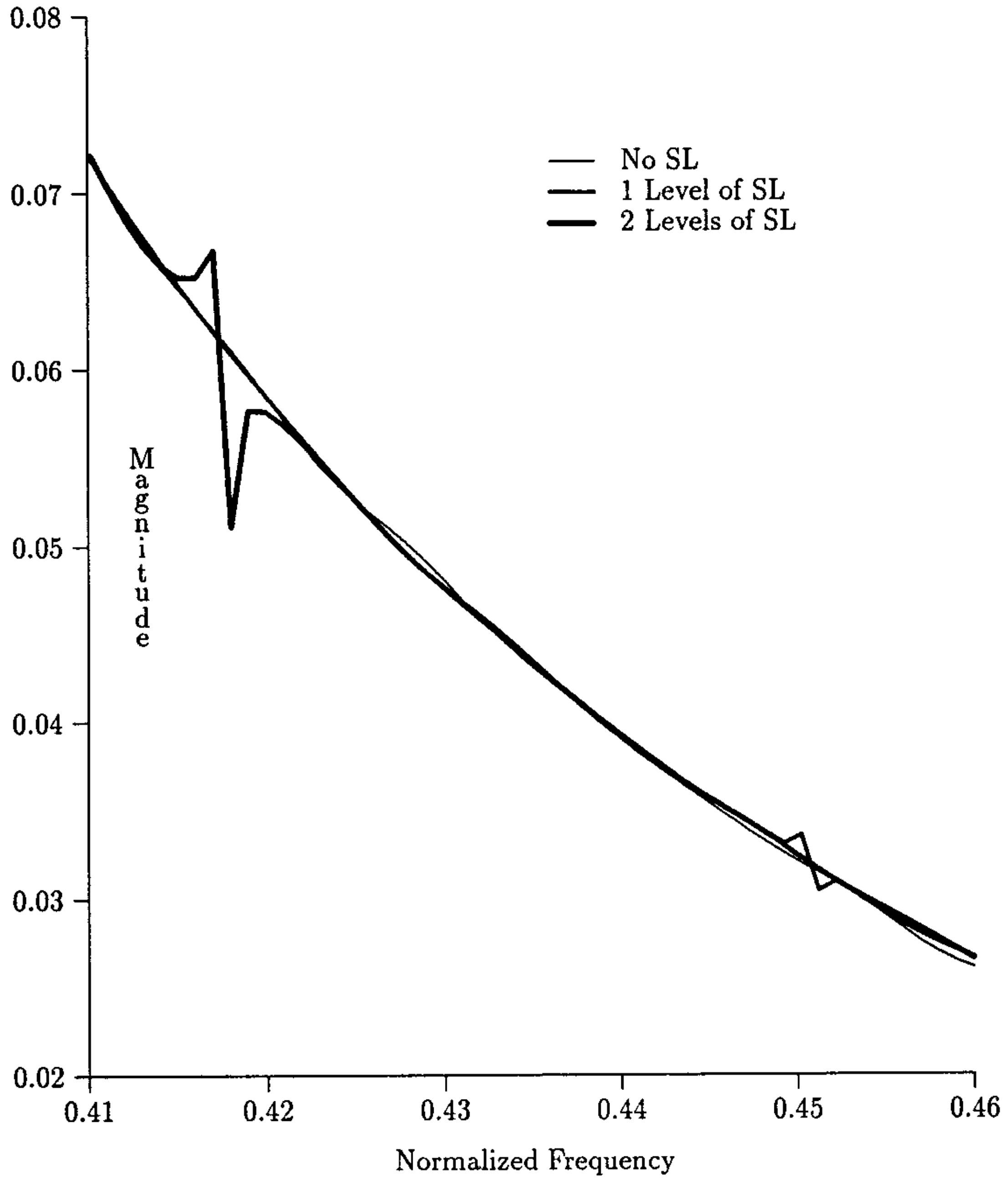


Figure B.4: Region C of Frequency Response of F1 ($a = -1.09$, $b = -0.90$)

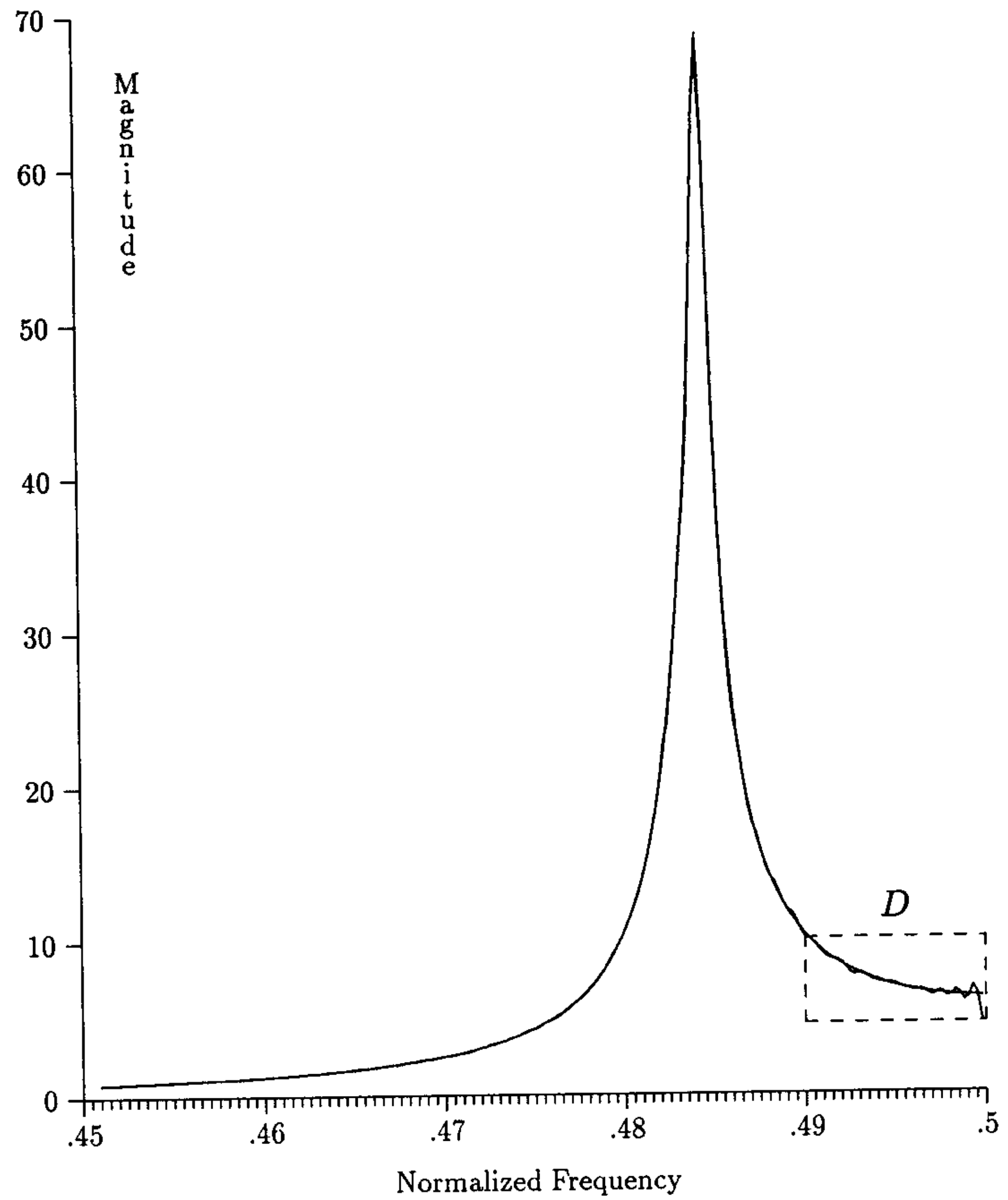


Figure B.5: Frequency Response of F2 ($a = -1.98$, $b = -0.99$)

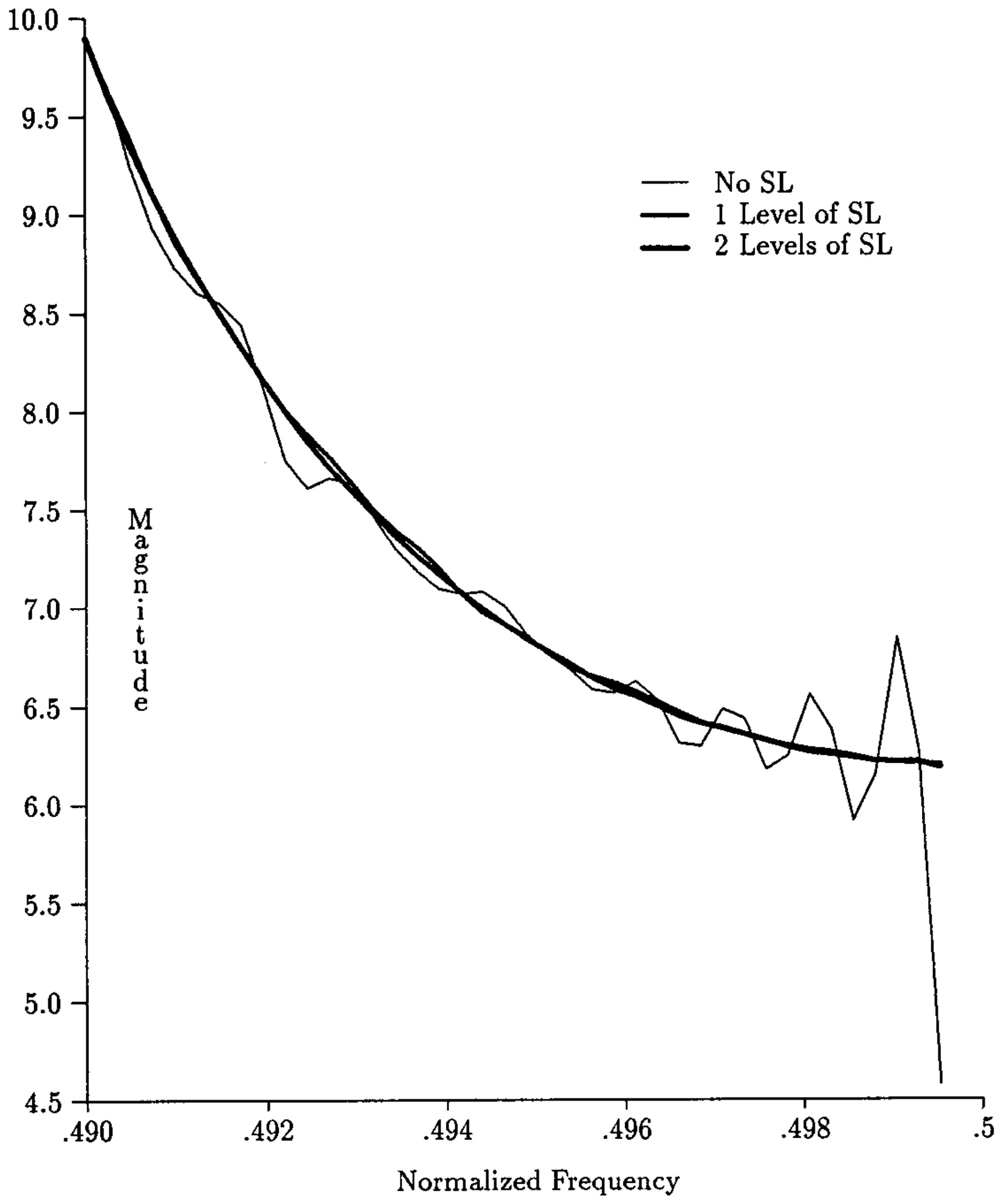


Figure B.6: Region D of Frequency Response of F2 ($a = -1.98$, $b = -0.99$)

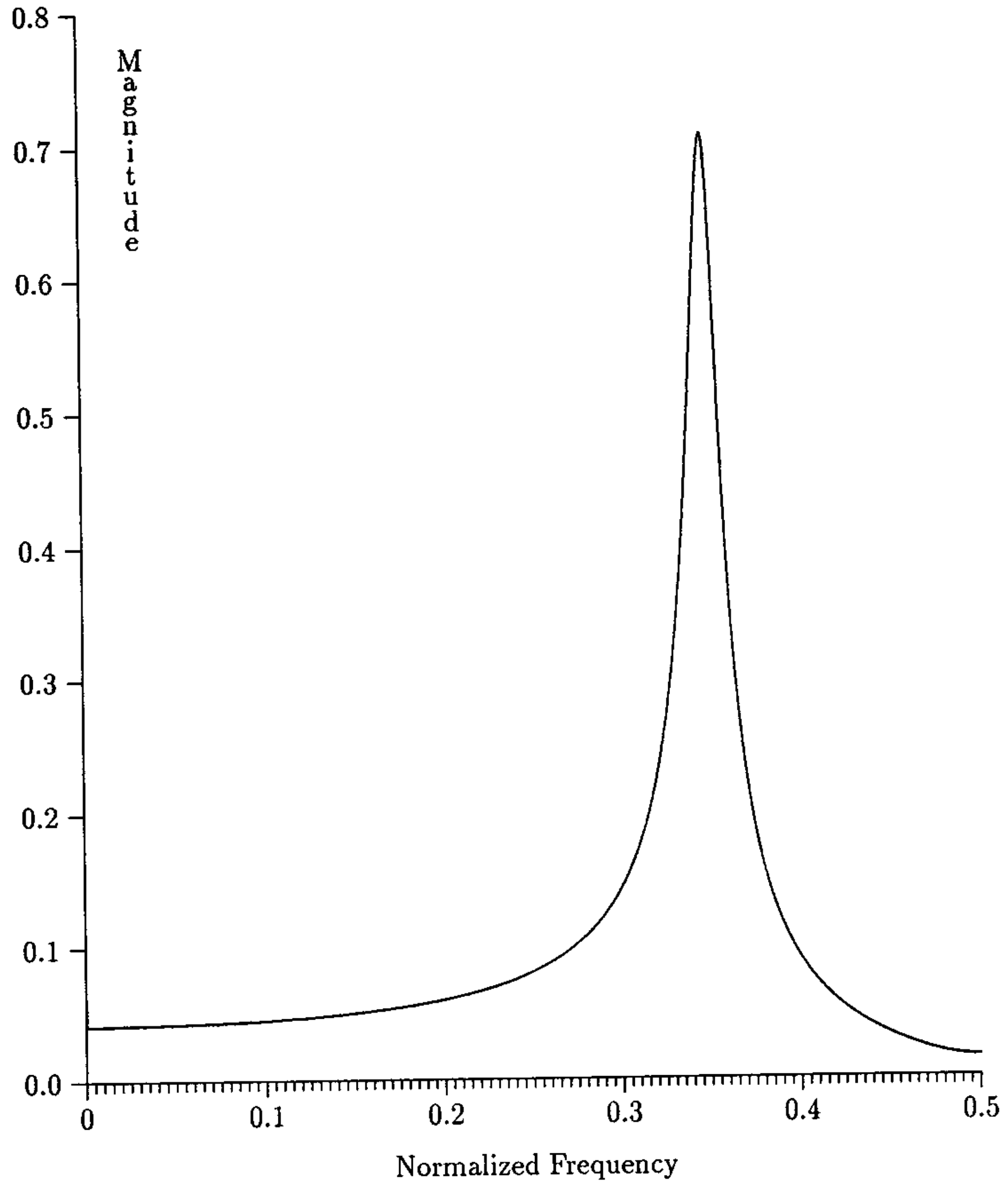


Figure B.7: Frequency Response of F1 with Dynamic Scaling ($a = -1.09$, $b = -0.90$)

REFERENCES

[Av61] Avizienis, A., *Signed-Digit Number Representation for Fast Parallel Arithmetic*, IEEE Trans. Electronic Computers, Vol. EC-10, pp. 389–400, 1961.

[BaWn92] P.H. Baur and Jie Wang, *The Effect of Various Floating Point Formats On The Absolute Error Bound In Recursive Filtering*, Proc. 1992 Int. Conf. on Acoustics, Speech and Signal Processing, vol. 4, pp. 413–416.

[BlCh92] W. Bliss and K. Chang, *Roundoff and Coefficient Quantization Noise of Pipelined Scattered-Look-Ahead Filters with Decomposition*, Proc. 1992 Int. Conf. on Acoustics, Speech and Signal Processing, vol. 4, pp. 433–436.

[BlCh91] W.G. Bliss and K.-H. Chang, *Roundoff Noise of Pipelined Scattered Look-ahead IIR Digital Filters*, Proc. 25th Annual Asilomar Conference on Signals, Systems and Computers, Vol. 2, pp. 1026–1030, 1991.

[Bra89a] Ralph H. Brackert, Jr., M.D. Ercegovac, and Alan N. Wilson, Jr., *Design of an On-Line Multiply-Add Module for Recursive Digital Filters*, Proc. of 9th Symposium on Computer Arithmetic, pp. 34–41, 1989.

[Bra89b] Ralph H. Brackert, Jr., *Design and Implementation of a High-Speed Recursive Digital Filter Using On-Line Arithmetic*, Ph.D. dissertation, University of California, Los Angeles, 1989.

[Cha91] Lloyd C. Cha, *A Recursive Digital Filter Using On-Line Arithmetic*, Masters thesis, University of California, Los Angeles, 1991.

[DSP2] Edited by Digital Signal Processing Committee, IEEE Acoustics, Speech and Signal Processing Society, *Selected Papers in Digital Signal Processing, II,* IEEE Press, pp. 412-451, 1976.

[Erc84] Miloš D. Ercegovac, *On-line Arithmetic: An Overview* SPIE Vol. 495 Real-Time Signal Processing VII, pp. 86–93, 1984.

[Erc77] Miloš D. Ercegovac, *A General Hardware-Oriented Method for Evaluation of Functions and Computations in a Digital Computer*, IEEE Transactions on Computers, C-26(7), pp. 667–680, July 1977.

[ErLn87] Miloš D. Ercegovac and Tomas Lang, *Radix-4 Multiplication without Carry-Propagate Addition*, Proc. of The International Conf. on Computer Design: VLSI in Computers & Processors, 1987.

[ErLn88] Miloš D. Ercegovac and Tomas Lang, *On-Line Arithmetic: A Design Methodology and Applications*, VLSI Signal Processing III, IEEE Press, R.W. Broderson and H.S. Moscovitz Eds., pp. 252–263, 1988.

[ErLn89] Miloš D. Ercegovac and Tomas Lang, *Most-Significant-Digit-First and On-Line Arithmetic Approaches for the Design of Recursive Filters*, Proc. 23rd Annual Asilomar Conference on Signals, Systems and Computers, pp. 7–11, 1989.

[ErLn92] Miloš D. Ercegovac and Tomas Lang, *Fast Arithmetic for Recursive Computations*, VLSI Signal Processing V, IEEE Press, K. Yao, R. Jain, W. Przytula, J. Rabaey Eds., pp. 14–28, 1992.

[FnEr92a] John S. Fernando and Miloš D. Ercegovac, *On-Line Arithmetic Modules for Recursive Digital Filters*, Proc. 26th Annual Asilomar Conference on Signals, Systems and Computers, pp. 681–685, 1992.

[FnEr92b] John S. Fernando and Miloš D. Ercegovac, *Conventional and On-Line Arithmetic Designs For High-Speed Recursive Digital Filters*, VLSI Signal Processing V, IEEE Press, K. Yao, R. Jain, W. Przytula, J. Rabaey Eds., pp. 81–90, 1992.

[GrTn92] P.J. Graumann, L.E. Turner, *Implementing Digital Signal Processing Algorithms using Pipelined Bit-Serial Arithmetic and Field Programmable Gate Arrays*, Proc. 1992 ACM First International Workshop on Field-Programmable Gate Arrays, Feb. 1992.

[HDSH86] K. Hayashi, K. Dhar, K. Sugahara, K. Hirano, *Design of High-Speed Digital Filters Suitable for Multi-DSP Implementation*, IEEE Trans. on Circuits and Systems, Vol. CAS-33, No. 2, pp. 202–217, Feb. 1986.

[KlNo88] U. Kleine and T.G. Noll, *Wave Digital Filters Using Carry-Save Arithmetic*, Proc. 1988 IEEE International Symposium on Circuits and Systems, pp. 1757–1762.

[KnMc88] Simon C. Knowles, John G. McWhirter, *An Improved Bit-Level Systolic Architecture For IIR Filtering*, Proc. 1988 Int. Conf. on Systolic Arrays, pp. 205–214.

[KnMc89] S.C. Knowles, J.G. McWhirter, R.F. Woods, J.V. McCanny, *Bit-Level Systolic Architectures for High Performance IIR Filtering*, Journal of VLSI Signal Processing, Kluwer Academic Publishers, Vol. 1, No. 1, August 1989, pp 9–24.

[MnMc90] O.C. McNally, J.V. McCanny, R.F. Woods, *Optimised Bit Level Architectures For IIR Filtering*, Proc. 1990 IEEE International Conference on Computer Design: VLSI in Computers & Processors, pp. 302–306.

[KnMc90] R.F. Woods, J.V. McCanny, S.C. Knowles, O.C. McNally, *A High Performance IIR Digital Filter Chip*, Proc. 1990 IEEE International Symposium on Circuits and Systems, Vol. 2, pp. 1410–1414.

[KnMc91] O.C. McNally, J.V. McCanny, R.F. Woods, *A 40 Megasample IIR Filter Chip*, Proc. International Conference on Application Specific Array Processors, IEEE Press, pp. 416–430, 1991.

[Kog73] P.M.Kogge and H.S. Stone, *A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations*, IEEE Trans. on Computers, Vol. C-22, No.8, pp. 786–793, August 1973.

[Kog81] Kogge Peter M., *The Architecture of Pipelined Computers*, McGraw-Hill, 1981.

[KRD92] Janusz Konrad, Jan Radecki and Eric Dubois, *On The Design of Finite Wordlength IIR Filters for Video Applications*, Proc. 1992 Int. Conf. on Acoustics, Speech and Signal Processing, vol. 4, pp. 341–344.

[LFH90] Marcel Lapointe, Paul Fortier, and Huu Tuê Huynh, *A New Faster and Simpler Systolic Structure For IIR Filters*, Proc. 1990 IEEE International Symposium on Circuits and Systems, Vol. 2, pp. 1227–1231.

[LnTr73] James L. Long and Timothy N. Trick, *An Absolute Bound on Limit Cycles Due to Roundoff Errors in Digital Filters*, IEEE Trans. Audio and Electroacoust., vol. AU-21, pp. 27–30, Feb. 1973.

[LSI86] LSI Logic Corporation, *Macrocells, Macrofunctions Databook and Design Manual*, Oct. 1986.

[LSI87] LSI Logic Corporation, *1.5-Micron Compacted Array^{TM} Technology Databook*, July 1987.

[LSI92] LSI Logic Corporation, *0.7-Micron Array-Based Products Databook*, April 1992.

[Opp70] Alan V. Oppenheim, *Realization of Digital Filters Using Block-Floating-Point Arithmetic*, IEEE Trans. Audio and Electroacoustics, vol. AU-18, pp. 130–136, June 1970.

[HtPr92] Mehdi Hatamian and K. K. Parhi, *An 85MHz Fourth-Order Programmable IIR Digital Filter Chip*, IEEE Journal of Solid-State Circuits, vol. 27, No. 2, pp. 175–183, Feb. 1992.

[MtLw81] D. Mitra and V.B. Lawrence, *Controlled Rounding Arithmetics, for Second-Order Direct-Form Digital Filters, that Eliminate All Self-Sustaining Oscillations*, IEEE Trans. on Circuits and Systems, Vol. CAS-28, pp. 894–905, Sept. 1981.

[PrHt88] Keshab K. Parhi and Mehdi Hatamian, *A High Sample Rate Recursive Digital Filter Chip*, Chapter 1, Part 1, VLSI Signal Processing, III, IEEE Press, pp. 3–14, 1988.

[PrMn88] John G. Proakis and Dimitris G. Manolakis, *Introduction to Digital Signal Processing,* Macmillan, 1988.

[Ms88] D.G. Messerschmitt, *Breaking the Recursive Bottleneck*, Performance Limits in Communication Theory and Practice, NATO ASI Series, Vol. 142, J.K. Skwirzynski, Ed., Kluwer Academic Publishers, pp. 3–19, 1988.

[PrMs87a] Keshab Kumar Parhi and David G. Messerschmitt, *Look-Ahead Computation: Improving the Iteration Bound in Linear Recursions,* Proc. 1987 Int. Conf. on Acoustics, Speech and Signal Processing, vol. 3, pp. 1855–1858.

[PrMs87b] Keshab Kumar Parhi and David G. Messerschmitt, *Concurrent Cellular VLSI Adaptive Filter Architectures*, IEEE Trans. on Circuits and Systems, Vol. CAS-34, No.10, pp. 1141–1150, October 1987.

[Sam88] H. Samueli, Thu-ji Lin and Willen S. Lao, *A Comparison of Recursive Digital Filter Structures Suitable For High-Speed Custom VLSI*

Implementation, Proc. 21st Asilomar Conference on Signals, Systems and Computers, pp. 23–27.

[SnKs72] J.W. Sandberg and J.F. Kaiser, *A Bound on Limit Cycles in Fixed-Point Implementations of Digital Filters*, IEEE Trans. Audio and Electroacoustics, vol. AU-20, pp. 110–112, June 1972.

[TrEc77] Kishor S. Trivedi and Miloš D. Ercegovac, *On-line Algorithms for Division and Multiplication* IEEE Transactions on Computers, C-26(7), pp. 681–687, July 1977.

[Tu90] Paul K.-G. Tu, *On-Line Arithmetic Algorithms for Efficient Implementations*, Ph.D. Thesis, University of California, Los Angeles 1990.

[TuEr91] Paul K.-G. Tu and Miloš D. Ercegovac, *Application of On-Line Arithmetic Algorithms to SVD Computation: Preliminary Results*, Proc. of 10th IEEE Symposium on Computer Arithmetic, pp. 246–255, June 1991.

[UnAb75] Z. Unver and K. Abdullah, *A Tighter Practical Bound on Quantization Errors in Second-Order Digital Filters With Complex Conjugate Poles*, IEEE Trans. Circuits and Systems, vol. CAS-22, pp. 632–633, July 1975.